Titre: Title:	Utilisation de langages formels pour la modélisation et la résolution de problèmes de planification de quarts de travail
Auteur: Author:	Marie-Claude Côté
Date:	2010
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Côté, M.-C. (2010). Utilisation de langages formels pour la modélisation et la résolution de problèmes de planification de quarts de travail [Thèse de doctorat, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/422/

UNIVERSITÉ DE MONTRÉAL

UTILISATION DE LANGAGES FORMELS POUR LA MODÉLISATION ET LA RÉSOLUTION DE PROBLÈMES DE PLANIFICATION DE QUARTS DE TRAVAIL

MARIE-CLAUDE CÔTÉ
DÉPARTEMENT DE MATHÉMATIQUES ET DE GÉNIE INDUSTRIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

THÈSE PRÉSENTÉE EN VUE DE L'OBTENTION DU DIPLÔME DE
PHILOSOPHIÆ DOCTOR
(MATHÉMATIQUES)
JUILLET 2010

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Cette thèse intitulée :

UTILISATION DE LANGAGES FORMELS POUR LA MODÉLISATION ET LA RÉSOLUTION DE PROBLÈMES DE PLANIFICATION DE QUARTS DE TRAVAIL

présentée par : Mme. CÔTÉ Marie-Claude.
en vue de l'obtention du diplôme de : Philosophiæ Doctor
a été dûment acceptée par le jury constitué de :

M. DESAULNIERS Guy, Ph.D., président.
M. ROUSSEAU Louis-Martin, Ph.D., membre et directeur de recherche.
M. GENDRON Bernard, Ph.D., membre et co-directeur de recherche.
Mme. REKIK Monia, Ph.D., membre externe.
M. GAMACHE Michel, Ph.D., membre.

À Pierre-Marc, Édouard et Charles.

Remerciements

Je voudrais premièrement remercier mes directeurs Louis-Martin Rousseau et Bernard Gendron pour leur encadrement pendant ces dernières années. Votre disponibilité et votre confiance ont été très appréciées. Merci pour votre motivation à l'égard du projet de cette thèse et de votre grand support pour m'aider à le mener à terme. Je tiens aussi à vous remercier pour m'avoir encouragé à passer au doctorat, c'est une décision que je n'ai jamais regrettée. J'ai beaucoup apprécié travailler avec vous pendant ces années.

Je suis très reconnaissante à Mme Monia Rekik, M. Guy Desaulniers et M. Michel Gamache pour avoir accepté de siéger sur mon jury.

Je souhaite aussi remercier tous mes collègues du laboratoire de programmation par contraintes. En particulier, merci à Simon, Alessandro et Claude-Guy pour votre support, votre aide et votre amitié.

Merci à Marc Brisson pour son aide au tout début de cette thèse. Merci également à Serge Bisaillon et François Guertin pour votre support et votre respect concernant tous mes nombreux questionnements techniques lors du développement de la dernière partie de ce travail.

Finalement, je désire remercier ma famille pour leurs encouragements et leur confiance. Tout particulièrement, Pierre-Marc, merci pour tout et merci Édouard, tu as été la meilleure motivation pour bien m'organiser et bien travailler.

Résumé

La planification d'horaires de personnel représente un défi pour plusieurs organisations. Dans cette thèse, nous étudions différentes variantes du problème de planification des quarts de travail. La planification des quarts de travail représente la sélection d'un ensemble de quarts couvrant une période de temps, typiquement une journée à une semaine, divisée en périodes de durées égales pour lesquelles un nombre d'employés requis est donné. Un quart est défini par son heure de début, sa durée et par sa composition en termes d'affectation de pauses et d'activités de travail.

Nous divisons les problèmes de planification de quarts en quatre classes. Premièrement, les problèmes où les employés sont considérés comme identiques et les problèmes où les employés ont des caractéristiques individuelles qui les distinguent et qui doivent être prises en compte dans la sélection des quarts. Nous référons à la première classe de problèmes comme étant des problèmes anonymes et à la seconde, comme étant des problèmes personnalisés. Ensuite, nous différencions les problèmes de planification de quarts mono-activité et multi-activités. La première classe de problèmes est en fait un cas particulier de la deuxième. Elle détermine quelles périodes du quart sont affectées à des activités de travail et quelles périodes sont affectées à des activités de repos (repos, pause, repas), alors que les problèmes de planification de quarts multi-activités définissent pour chaque période de travail du quart quelle activité de travail, parmi un ensemble d'activités de travail pouvant être effectuées, lui est affectée. Dans ce cas, pour chaque activité de travail et pour chaque période de temps sur lequel s'étend l'horaire, le nombre d'employés requis est donné.

Dans cette thèse, nous abordons chacune des classes de problèmes de planification de quarts à l'aide de différentes approches, toutes basées sur la formulation des contraintes restreignant la formation des quarts par des outils issus de la théorie des langages formels. En effet, à l'aide d'automates et de grammaires, nous pouvons définir des langages qui sont composés d'un ensemble de mots représentant des quarts respectant les contraintes de notre problème. Les automates et les grammaires sont des outils très expressifs qui permettent de formuler plusieurs concepts relatifs à notre contexte de manière relativement naturelle.

Plus précisément, notre première contribution présente comment, à partir d'un automate ou d'une grammaire définissant les règles régissant la composition des quarts pour un problème, nous pouvons générer automatiquement un modèle linéaire en nombres entiers basé sur des variables d'affectation binaires dans une structure de graphe qui encode tous les quarts permis par ce problème. La transformation d'un automate ou d'une grammaire en modèle de programmation mathématique est inspirée de structures de la programmation par contraintes. Un exemple expérimental de problème de planification de quarts multi-activités montre la puissance de la modélisation utilisant les langages formels. En effet, cette nouvelle approche de modélisation permet d'aborder des règles très complexes en ce qui concerne la composition des quarts multi-activités. Malgré que l'intérêt de ce type de modèles dans le contexte de problèmes de planification de quarts soit indéniable, les résultats expérimentaux soulèvent les limites de la formulation décomposée par employés et définie avec des variables d'affectation binaires. En effet, avec le nombre d'employés et le nombre d'activités de travail qui augmentent, le modèle devient difficile à résoudre directement.

Pour pallier la croissance des modèles et les problèmes reliés à la symmétrie venant des employés identiques, notre seconde contribution est un modèle implicite basée sur une grammaire permettant de modéliser les problèmes de planification de quarts multi-activités anonymes. À partir d'une grammaire qui encode toutes les règles sur la composition des quarts, un graphe semblable à celui défini dans le premier article est généré. Plutôt que d'utiliser des variables d'affectation binaires associées aux quarts admissibles pour chacun des employés, le modèle utilise des variables entières non négatives pour représenter implicitement dans un seul modèle tous les quarts permis par le problème. La solution optimale de ce modèle mathématique en nombres entiers donne le nombre de quarts requis pour couvrir les demandes périodiques et le nombre de quarts affectés à chaque activité de travail à chaque période. Une procédure doit être appliquée à la solution optimale implicite pour obtenir les quarts explicites. Les résultats expérimentaux montrent que les modèles générés par notre approche sont faciles à résoudre par un logiciel commercial, en particulier sur des instances couvrant une journée de 96 périodes et jusqu'à 10 activités de travail. De plus, nous montrons qu'ils engendrent le même saut d'intégrité que des modèles reconnus dans

la littérature. À notre connaissance, il s'agit de la première méthode implicite pouvant aborder les problèmes de planification de quarts multi-activités.

La dernière contribution présente une méthode de génération de colonnes aussi basée sur les grammaires pour permettre d'aborder efficacement les problèmes de planification de quarts multi-activités personnalisés. La méthode utilise un modèle de recouvrement d'ensembles pour modéliser le problème et la relaxation linéaire du problème est résolue par une méthode de génération de colonnes basée sur un sous-problème reposant sur un graphe généré à partir d'une grammaire. Un sous-problème par employé est utilisé pour modéliser l'ensemble des quarts que l'employé peut effectuer compte tenu de ses caractéristiques personnelles. Les sous-problèmes sont résolus par un algorithme de programmation dynamique. Une méthode de branchement adaptée permet de converger vers des solutions entières et de résoudre les problèmes de manière exacte. Cette méthode fait intervenir un nouveau sous-problème de génération de colonnes ainsi qu'une nouvelle méthode de branchement pour ce type de problème. L'approche de modélisation proposée est très flexible et contrairement aux rares approches existantes dans la littérature, elle permet de résoudre exactement différentes versions du problème de planification de quarts multi-activités personnalisés.

Abstract

Personnel scheduling is a challenging problem for many organizations. In this thesis, we address different versions of the shift scheduling problem. The shift scheduling problem is to select a set of shifts to cover a planning horizon, typically from 1 to 7 days, divided into periods of equal length for which the required numbers of employees are given. A shift is defined by its starting time, its duration and its composition. Its composition is defined by the position of the breaks and work activities within the shift.

We divide the shift scheduling problems into four main classes. First, we distinguish the problems where the employees are considered to be identical from the problems where each employee has individual characteristics that must be taken into account when assigning them to shifts. We call the first class the anonymous problems and the second one, the personalized problems. Then, we differenciate two other classes of problems, the mono-activity and the multi-activity shift scheduling problems. The former is a particular case of the latter and consists in specifying the work and rest periods to assign to the shifts. In the multi-activity case, we are also interested in a set of distinct work-activities to be performed. So, not only should we specify if the shift is assigned to a work-activity or not at a given period, but we must specify to which work-activity it is assigned. In this case, for each work-activity and each period, the required number of employees is given.

In this thesis, we study each of these classes of shift scheduling problems with different approaches where constraints on shift construction are formulated with tools based on formal languages. In fact, using automata and grammars, we define languages composed of words that represent allowed shifts for our problem.

More precisely, our first contribution presents how, from a finite automata or a context-free grammar defining the rules constraining the construction of shifts for a given problem, one can automatically generate an integer programming model based on binary assignment variables in a graph structure embedding every allowed shift for this problem. The transformation of an automaton or a grammar into a mathematical programming model is inspired by structures from constraint programming. Experi-

ments on a multi-activity shift scheduling problem show that formal language based modeling is very powerful and allows us to address complex rules in the construction of multi-activity shifts. However, while the relevance of our modeling approach is clear, the experimental results reveal some limitations on the scalability of the formulation based on binary assignment variables and decomposed on employees. In fact, when the number of employees and the number of work-activities grow, the model is hard to solve directly.

To address both the growing size of the models and the symmetry issues arising from a context where employees are identical, our second contribution is an implicit model based on grammars allowing us to model and solve anonymous multi-activity shift scheduling problems efficiently. From a grammar encoding every restriction in the composition of shifts, we use the graph structure presented in the first article. Instead of using binary variables associated with allowed shifts for each employee, we use nonnegative integer variables to implicitly represent, in a single graph structure, every allowed shift for the problem. The optimal solution to the corresponding integer programming model gives the required number of shifts to cover the demands of the problem and the number of shifts assigned to each work-activity at each period. From these informations, a procedure extracts the optimal explicit solution from the implicit one. Experimental results show that this modeling approach gives easy-to-solve models and can address a wide variety of rules over shifts. We also prove that the models obtained with our technique yield the same integrality gap as well known models in the literature. To our knowledge, this is the first implicit method that can address the multi-activity shift scheduling cases.

Our last contribution is a column generation approach also based on grammars that allows us to successfully solve personalized multi-activity shift scheduling problems. The method uses a set covering formulation to model the problem and solves the linear relaxation with a column generation method based on a pricing subproblem using the graph generated from a grammar. A subproblem is defined for each available employee and represents the set of shifts that the given employee is allowed to perform given his individual characteristics. The subproblems are solved with a dynamic programming algorithm. An adapted branching procedure is proposed to find integer solutions and solve the problem exactly. Our method uses a new col-

umn generation subproblem and a new branching strategy for this class of problems. The modeling approach is very flexible and contrary to the few other approaches proposed in the literature, it can solve exactly different versions of the personalized multi-activity shift scheduling problem.

Table des matières

Dédicace iii

Remerciements iv

Résumé v

Abstract viii

Table des matières xi

Liste des tableaux xv

Liste des figures xvi

Liste des annexes et des algorithmes xvii

Chapitre 1 INTRODUCTION 1
1.1 Définitions et concepts de base 1
1.2 Éléments de la problématique 3
1.3 Objectifs de recherche 5
1.4 Plan de la thèse 5

Chapitre 2 REVUE DE LITTÉRATURE 7
2.1 Planification d'horaires de personnel 7
2.2 Modèles et méthodes de programmation mathématique 7
2.2.1 Problèmes de planification de quarts mono-activité 13
2.2.2 Problèmes de planification d'horaires de personnel multi-activités 17
2.2.3 Approches basées sur des réseaux pour les problèmes de planification d'horaires de personnel 19
2.2.4 Génération de colonnes et branch-and-price 22

2.3 Programmation par contraintes et langages formels 25
2.3.1 Modélisation par programmation par contraintes 25
2.3.2 Langages formels appliqués à la programmation par contraintes 25
2.3.3 Définition d'automates et de grammaires pour les problèmes de planification de quarts de travail 32
2.3.4 Langages formels, programmation par contraintes et planification de quarts de travail 35

Chapitre 3 DÉMARCHE DE L'ENSEMBLE DU TRAVAIL DE RECHERCHE ET ORGANISATION GÉNÉRALE DU DOCUMENT 38

Chapitre 4 FORMAL LANGUAGES FOR INTEGER PROGRAMMING MODELING OF SHIFT SCHEDULING PROBLEMS 40
4.1 Introduction 43
4.2 Shift Scheduling Problems 44
4.3 Background Material 45
4.3.1 Automata and the CP Regular Constraint 45
4.3.2 Context-free Grammars and the CP Grammar Constraint 47
4.4 MIP Regular 49
4.5 MIP Grammar 54
4.6 Comparison Between MIP Regular and MIP Grammar 56
4.7 Case Study 59
4.7.1 Problem Definition 59
4.7.2 A Compact Assignment MIP Model 60
4.7.3 A MIP Regular Model 63
4.7.4 MIP Grammar Models 64
4.7.5 Computational Results 65
4.8 Conclusion 69

Chapitre 5 GRAMMAR-BASED INTEGER PROGRAMMING MODELS FOR MULTI-ACTIVITY SHIFT SCHEDULING 71
5.1 Introduction 74
5.2 Background Material 76

5.2.1 Shift Scheduling 76
5.2.2 Grammars 78
5.3 Grammar-Based Model for Shift Scheduling 84
5.4 Theoretical Properties of Grammar-Based Models 85
5.5 Computational Experiments 88
5.5.1 Shift Scheduling with Multiple Rest Breaks, Meal Breaks, and Break Windows 88
5.5.2 Shift Scheduling with Multiple Rest and Meal Breaks, and Multiple Work Activities 92
5.6 Conclusion 98

Chapitre 6 GRAMMAR-BASED COLUMN GENERATION FOR PERSONALIZED MULTI-ACTIVITY SHIFT SCHEDULING 100
6.1 Introduction 102
6.2 Background Material 104
6.2.1 Literature Review 104
6.2.2 Definitions 107
6.3 Grammar-Based Column Generation Approach 109
6.3.1 Restricted Master Problem 109
6.3.2 Pricing Subproblem 111
6.3.3 Branching Rule 113
6.4 Computational Experiments 114
6.4.1 Problem Instances From Demassey *et al* (2006) 115
6.4.2 Problem Instances From Lequy *et al* (2009) 121
6.4.3 Summary 127
6.5 Conclusion 128

Chapitre 7 DISCUSSION GÉNÉRALE 129

Chapitre 8 CONCLUSION 132
8.1 Synthèse des travaux 132
8.2 Limitations de la solution proposée 134
8.3 Améliorations et avenues de recherche futures 135

Références 136

Annexes 142

Liste des tableaux

Tableau 2.1	Dérivation du mot *aab* à partir de la grammaire G de l'exemple 3	29
Table 4.1	Results for the compact assignment MIP model	67
Table 4.2	Results for the `MIP regular` model	67
Table 4.3	Results for the partial `MIP grammar` model	68
Table 4.4	Results for the complete `MIP grammar` model	68
Table 4.5	Results for the two work activity instances	69
Table 5.1	Derivation of word *wwbw* from grammar G of Example 4 . . .	80
Table 5.2	Number of instances solved to optimality within the four- minute time limit	90
Table 5.3	Summary for the instances solved to optimality	91
Table 5.4	Model comparison on the one-activity problem	95
Table 5.5	Multi-activity problems with the Implicit Grammar model . .	96
Table 5.6	Comparison of solution times (seconds) between employee-based explicit formulations and the implicit grammar model on the one and two-activities instances to obtain a near-optimal solution ($Gap \leq 1\%$) in less than 3600 seconds	97
Table 6.1	Derivation of word $j_1bj_2j_2$ from grammar G of Example 5 . . .	109
Table 6.2	Comparison with Demassey *et al.* (2006) approach	119
Table 6.3	Comparison between approaches based on formal languages on instances with one and two work-activities-CPU time(s)	120
Table 6.4	Comparison with exact methods on the smallest instances of the first set of instances of Lequy *et al.* (2009) problem	124
Table 6.5	Comparison with the *Block model* to find the first integer solution on the largest instances of the first set of instances of Lequy *et al.* (2009) problem	124
Table 6.6	Comparison with the *Horizon CG* approach on the first set of instances of Lequy *et al.* (2009) problem	125
Table 6.7	Comparison with the *Horizon CG* approach on the second set of instances of Lequy *et al.* (2009) problem	126

Liste des figures

FIGURE 2.1 Automate Π où les états sont représentés par des cercles, les états finaux, par des cercles doubles et les transitions, par des arcs. 27

FIGURE 2.2 Graphe à couches construit par la procédure `initialiser`() pour `regular`($x_1, x_2, \ldots, x_5$, Π) 28

FIGURE 2.3 Arbres d'analyse pour les mots de longueur 3 reconnus par la grammaire G de l'exemple 3 30

FIGURE 2.4 GOA Γ associé à la grammaire de l'exemple 3 pour des mots de longueur 3 32

FIGURE 2.5 Automate pour le problème de planification de quarts de l'exemple 4 pour 2 activités de travail 34

FIGURE 4.1 DFA Π with each state shown as a circle, each final state as a double circle, and each transition as an arc. 46

FIGURE 4.2 And/or tree constructed by Algorithm 1 on the grammar of Example 2 and a sequence of length $n = 3$. 49

FIGURE 4.3 Automaton π_5 recognizing all sequences of length 5 on alphabet $\Sigma = \{a, b, c\}$ 50

FIGURE 4.4 Automaton $A = \Pi \cap \pi_5$ 51

FIGURE 4.5 Graph G associated to automaton A 51

FIGURE 4.6 The relation between the arc $(4, b, 2, 3)$ from the layered graph of `MIP regular` depicted in Figure 4.5 and the corresponding nodes in the and/or graph. The label of each node is written on the left and the associated 0-1 variable on the right. . . . 58

FIGURE 4.7 DFA Π_2 for two activities 64

FIGURE 5.1 Parse trees for grammar G from Example 4 on words of length 4. 81

FIGURE 5.2 DAG Γ for grammar from Example 4 on a word of length 4. . 83

FIGURE 6.1 Parse tree for word $j_1 b j_2 j_2$ derived from grammar G of Example 5110

FIGURE 6.2 DAG for grammar G from Example 5 on words of length 4 . . 111

Liste des annexes et des algorithmes

1 Construction of a graph embedding all possible parsing trees of sequences of length n recognized by the grammar G 48

Annexe A Schedule Construction 142
2 Extracting detailed schedules from an implicit grammar solution . . . 142

Chapitre 1

INTRODUCTION

La création d'horaires de personnel est un problème très important, mais difficile pour plusieurs types d'organisations de petites et grandes tailles. Les conventions collectives complexes, les considérations ergonomiques, les préférences, les compétences et la disponibilité du personnel, les différentes localisations de travail et les questions monétaires sont autant d'aspects qui rendent la conception des horaires difficile. Tenir compte de plusieurs de ces aspects simultanément rend presque impossible la tâche de créer manuellement un horaire optimal. C'est pourquoi l'utilisation de ressources informatiques et de modèles mathématiques adaptés est nécessaire dans plusieurs contextes. Dans ce qui suit, nous introduirons d'abord le problème que nous allons aborder dans cette thèse, pour ensuite présenter les détails de la problématique et préciser nos objectifs.

1.1 Définitions et concepts de base

Un *horaire de personnel complet* peut être représenté comme un ensemble d'horaires de travail individuels. Un *horaire de travail individuel* est divisé en *périodes* de durées égales (par exemple, 15 minutes, 30 minutes, 1 journée). Les périodes sont des subvisions de l'*horizon de planification* qui est l'intervalle de temps pour lequel l'horaire est requis. Pour chaque employé et chaque période est affectée une activité. Une *activité* peut être un type de quart (jour, soir, nuit, congé) ou faire partie d'un quart (pause, repas, repos, travail, activité de travail spécifique). L'affectation des activités aux périodes d'un horaire de travail individuel doit respecter un ensemble de règles, généralement issues des conventions collectives en vigueur. Finalement, l'ensemble des horaires de travail individuels doit former un horaire complet qui respecte le minimum (et dans certains cas, le maximum) d'employés requis pour compléter chaque activité à chaque période.

Dans le présent travail, nous nous intéressons à la *planification de quarts de travail*

sur des horizons de planification de 1 à 7 jours divisés en périodes de 15 minutes. Sur les horizons de planification de plus d'une journée, plusieurs quarts de travail peuvent être affectés à un même employé. Un *quart de travail* est défini par une période de début, une longueur et sa composition. Dans certains travaux, on utilise l'appellation *type de quart* pour se référer à une combinaison début et longueur de quart, sans tenir compte de sa composition. En ce qui concerne la composition des quarts, nous faisons ici une distinction entre deux catégories de problèmes : les problèmes mono-activité et les problèmes multi-activités.

Les *problèmes de planification de quarts mono-activité* définissent pour chaque quart de travail la période de début du quart, sa longueur et les périodes où l'employé est en pause. Les *problèmes de planification de quarts multi-activités* définissent, en plus, à quelle *activité de travail* l'employé est affecté à chaque période du quart où il n'est pas en pause. Les problèmes multi-activités sont des problèmes plus complexes à modéliser et à résoudre, notamment parce qu'ils peuvent faire apparaître de nouvelles règles contraignant la composition des quarts, en particulier en ce qui concerne la transition entre les activités de travail, le nombre de périodes consécutives pouvant être affectées à une même activité de travail et les compétences des employés à travailler sur les différentes activités de travail.

La question des compétences des employés amène une autre subdivision dans le problème de planification de quarts de travail : les problèmes anonymes et les problèmes personnalisés. Les *problèmes de planification de quarts anonymes* considèrent tous les employés comme identiques alors que les *problèmes de planification de quarts personnalisés* considèrent les caractéristiques de chaque employé, comme par exemple ses périodes de disponibilité ou dans le cas de problèmes multi-activités, ses compétences, le restreignant à un sous-ensemble des activités de travail devant être exécutées. Cette catégorie de problèmes peut aussi aborder les cas où une organisation a différents types d'employés, comme des employés à temps partiel et à temps plein qui ne sont pas régis par les mêmes règles de convention collective. Les problèmes personnalisés sont habituellement de grande taille dû au fait que chaque employé doit être représenté explicitement.

Dans la suite du document, nous utiliserons les termes "quart de travail" et "quart" comme des synonymes.

1.2 Éléments de la problématique

Les différentes sous-classes de problèmes de planification de quarts de travail énoncées dans la section 1.1 comportent des caractéristiques qui rendent leur formulation en modèles mathématiques et leur résolution parfois difficiles. Dans la présente section, nous détaillerons les différents défis que représentent les problèmes de planification de quarts de travail et nous soulèverons les points qui sont particuliers à chaque sous-classe de problème.

La définition d'un quart de travail peut être sujette à un certain nombre de contraintes. En voici quelques unes.

- *Contraintes sur les périodes où un quart peut débuter.* Par exemple, même si l'horizon de planification est divisé en période de 15 minutes, un quart peut être contraint de commencer sur l'heure ou la demi-heure. Aussi, il peut débuter pendant une fenêtre de temps donnée, par exemple, un quart peut devoir commencer entre 7h30 et 10h le matin.
- *Contraintes sur la longueur d'un quart.* Les quarts peuvent être contraints à une durée fixe, à une durée entre certaines bornes ou à différentes durées fixes. Ces durées peuvent différer si le quart appartient à une classe particulière (quart à temps partiel, quart à temps plein).
- *Contraintes sur le placement des pauses dans le quart.* Le placement des pauses dans un quart peut dépendre de son début et de sa longueur. Elles peuvent être restreintes à être placées à l'intérieur d'une fenêtre de temps prédéterminée selon le type de quart.
- *Contraintes sur le nombre de pauses.* Le nombre de pauses devant être affectées à un quart dépend souvent de la longueur du quart, du type de quart ou de la longueur des pauses.
- *Contraintes sur la durée des périodes de travail consécutives.* Dans certains environnements de planification, les séquences de périodes de travail et les pauses sont définies de manière à ce qu'une séquence de travail ait une durée entre certaines bornes données, i.e., qu'une pause (ou la fin du quart) ne peut pas être affectée si au moins un minimum de périodes de travail consécutives n'a pas été complété et une pause (ou la fin du quart) doit obligatoirement avoir lieu après une séquence de travail de longueur maximum.

Les problèmes de planification de quarts de travail multi-activités peuvent faire intervenir des contraintes additionnelles.

- *Contraintes sur la durée des périodes de travail consécutives sur une activité de travail donnée.* Les séquences de travail sur une même activité de travail peuvent être restreintes à une durée maximum et minimum.
- *Contraintes sur les enchaînements d'activités de travail.* Certaines activités de travail peuvent ne pas pouvoir être affectées consécutivement dans un quart de travail. Aussi, une activité de repos (pause ou repas) peut être requise entre deux activités de travail différentes.

Les problèmes de planification de quarts de travail personnalisés permettent d'aborder des aspects supplémentaires du problème.

- *Contraintes sur la disponibilité des employés.* La disponibilité d'un employé restreint les types de quarts qui peuvent lui être affectés.
- *Contraintes sur les compétences requises pour effectuer certaines activités de travail.* Dans le cas de problèmes de planification de quarts multi-activités, certaines compétences peuvent être requises pour qu'un employé puisse effectuer une activité de travail. Ces compétences doivent être prises en compte dans l'affectation des activités de travail au quart d'un employé.

Certains problèmes peuvent contenir un sous-ensemble de ces contraintes alors que d'autres peuvent toutes les faire intervenir. Trouver un horaire complet, satisfaisant les contraintes de demande sur le nombre d'employés requis à chaque période est donc souvent une tâche très difficile. Dans plusieurs cas, trouver un horaire complet ne suffit pas, l'objectif est d'identifier le meilleur horaire satisfaisant toutes les contraintes. La définition du meilleur horaire peut aussi varier grandement. Voici quelques critères :

- *Minimiser le coût global de l'horaire.* Étant donné qu'un coût est associé à chaque quart, l'objectif est de sélectionner l'ensemble des quarts formant un horaire complet réalisable engendrant le plus petit coût.
- *Minimiser le nombre d'employés nécessaires.* Dans les problèmes anonymes, l'objectif peut être de trouver un horaire réalisable nécessitant un nombre minimum d'employés.
- *Minimiser la sous-couverture de la demande.* Dans un contexte où on peut sous-couvrir la demande en nombre d'employés moyennant un coût par période

(et activité de travail, dans le cas multi-activités), l'objectif peut être de trouver l'horaire de travail de coût minimum en tenant compte des coûts de sous-couverture.
- *Minimiser le nombre de transitions entre activités de travail.* Dans un contexte multi-activités, une pénalité peut être attribuée lorsqu'un quart est affecté à plusieurs activités de travail différentes de manière à minimiser le va et vient des employés entre les activités de travail.

En somme, le nombre et la complexité des contraintes, ainsi que les différents objectifs font de la planification de quarts de travail et la conception d'un horaire de travail complet un problème très compliqué à modéliser et résoudre. De plus, dans les cas des quarts personnalisés et multi-activités, la taille du problème peut devenir très grande puisqu'elle dépend du nombre d'employés et du nombre d'activités de travail.

1.3 Objectifs de recherche

Dans cette recherche, nos objectifs sont, en premier lieu, de proposer des manières génériques de formuler les problèmes de planification de quarts de travail à l'aide de modèles de programmation mathématique de façon à pouvoir simultanément tenir en compte toutes les contraintes énumérées dans la section précédente et permettre une résolution des modèles à l'optimalité dans un temps raisonnable. Ensuite, nous désirons comparer ces différentes formulations de manière à identifier lesquelles sont les mieux adaptées dans les différents contextes. Nous souhaitons aussi comparer, lorsque c'est possible, nos différentes approches de modélisation à celles suggérées dans la littérature de manière théorique et empirique.

Finalement, nous voulons proposer une nouvelle formulation et une méthode de résolution permettant de résoudre de façon optimale la classe de problèmes la plus complexe et généralement de plus grande taille, soit les problèmes de planification de quarts multi-activités personnalisés.

1.4 Plan de la thèse

Au chapitre 2, nous présentons une revue de la littérature décrivant les principaux travaux qui abordent les différents aspects du problème de planification de quarts de

travail. Nous y introduisons aussi des notions nécessaires à la compréhension de la suite de la thèse. Le chapitre 3 décrit la démarche de l'ensemble du travail de recherche et l'organisation générale du document en mettant en relief la cohérence des objectifs de chaque partie par rapport aux objectifs globaux de la thèse. Le chapitre 4 présente deux formulations pour les problèmes de planification de quarts multi-activités personnalisés. Le chapitre 5 décrit et compare un nouveau modèle basé sur une des formulations introduites dans le chapitre 4 qui permet de modéliser et résoudre efficacement les problèmes anonymes. Dans le chapitre 6, nous introduisons une nouvelle formulation et une méthode de résolution permettant d'aborder les problèmes personnalisés. Au chapitre 7, nous faisons une discussion générale en regard de nos travaux en lien avec la revue de littérature. Finalement, le chapitre 8 conclut la thèse avec une synthèse, une critique et des améliorations à considérer concernant les travaux présentés.

Chapitre 2

REVUE DE LITTÉRATURE

2.1 Planification d'horaires de personnel

La planification d'horaires de personnel est un aspect très important pour plusieurs organisations et est souvent une tâche très complexe dûe à la taille des problèmes et au nombre parfois élevé de contraintes issues des différentes conventions collectives et objectifs en vigueur. Les différentes versions et parties du problème de planification d'horaires de personnel ont reçu beaucoup d'attention dans la littérature, comme en témoigne les revues de Ernst *et al.* (2004b) et Ernst *et al.* (2004a) qui recensent plus de 700 références. Les prochaines sections présentent les principales formulations et méthodes de résolution suggérées dans la littérature pour aborder les problèmes de planification de quarts de travail décrits dans le Chapitre 1.

2.2 Modèles et méthodes de programmation mathématique

Rappelons les quatre types de problèmes de planification de quarts présentés dans l'introduction : les problèmes de planification de quarts mono-activité, les problèmes de planification de quarts multi-activités, les problèmes de planification de quarts anonymes et les problèmes de planification de quarts personalisés. Ces problèmes comportent des caractéristiques très différentes et les méthodes adaptées à leur résolution diffèrent beaucoup d'un problème à l'autre.

Dantzig (1954) présente un modèle qui peut facilement s'adapter à tous les contextes décrits précédemment. L'idée est basée sur le fait que, pour un problème donné, on peut théoriquement énumérer tous les quarts respectant les contraintes du problème. Le modèle de Dantzig (1954) contient une variable par quart énuméré. Les contraintes suivantes présentent le modèle de Dantzig (1954) dans sa forme initiale.

$$Min \qquad \sum_{s \in \Omega} c_s x_s \tag{2.1}$$

$$\sum_{s \in \Omega} a_{is} x_s \geq d_i \qquad \forall\ i \in I, \tag{2.2}$$

$$x_s \geq 0\ et\ entier \quad \forall\ s \in \Omega. \tag{2.3}$$

où Ω est l'ensemble des quarts respectant les contraintes du problème, c_s est le coût associé au quart $s \in \Omega$, x_s est une variable de décision spécifiant le nombre d'employés affectés au quart $s \in \Omega$, I est l'ensemble des périodes de l'horizon de planification, $a_{is} = 1$ si la période $i \in I$ est affectée à une activité de travail dans le quart $s \in \Omega$ et d_i est le nombre minimum d'employés devant être affectés à une activité de travail à la période $i \in I$.

Puisque l'énumération de tous les quarts permis est en pratique impossible à faire pour plusieurs problèmes, d'autres approches ont été proposées. En particulier, la méthode de génération de colonnes (voir section 2.2.4) permet de résoudre la formulation de Dantzig (1954) en se basant sur le principe qu'une solution optimale peut être trouvée avec un sous-ensemble des variables du problème (colonnes).

Le modèle de Dantzig (1954) et les modèles basés sur la même idée (souvent identifiés comme des modèles de recouvrement) font partie des *modèles explicites* et nous dirons qu'il s'agit de modèles explicites basés sur les quarts puisque chacune de leurs variables représente un quart. Les modèles en *nombres entiers basés sur des variables d'affectation* forment une autre classe de modèles explicites, mais ceux-ci, basés sur les employés. Ils permettent typiquement de décrire les contraintes sur la composition des quarts avec des variables binaires x_{ei} spécifiant si un employé (ou un quart donné) e travaille ou non à la période i. Ces modèles sont souvent peu génériques et requièrent de bonnes connaissances en matière de modélisation mathématique. Des exemples de modèles en nombres entiers basés sur des variables d'affectation adressant les problèmes de planification d'horaires de personnel peuvent être trouvés dans Miller *et al.* (1976), Warner (1976), Beaulieu *et al.* (2000), Felici et Gentile (2004) et dans quelques formulations basées sur des réseaux (voir section 2.2.3). En théorie,

les modèles explicites peuvent aborder les problèmes les plus complexes comme les problèmes de planification de quarts multi-activités personnalisés. En pratique, ces modèles sont parfois trop complexes et leur tailles, trop grande pour être résolus directement. Dans ces cas, des méthodes comme la génération de colonnes ou des méthodes heuristiques adaptées sont parfois requises.

Une autre classe de modèles, les *modèles implicites*, a été développée pour la résolution de problèmes anonymes. Contrairement aux modèles explicites, où l'horaire de chaque employé peut être obtenu simplement en parcourant la solution optimale (i.e., dans un temps linéaire dans la taille du modèle), les modèles implicites utilisent un algorithme pour construire un horaire à partir de la solution implicite optimale. Cet algorithme est typiquement efficace, particulièrement en comparaison avec la résolution du modèle, mais de complexité généralement non linéaire dans la taille du modèle. Les formulations implicites permettent en général de diminuer le nombre de variables requis par les formulations explicites pour énumérer tous les quarts possibles. Les modèles présents dans la littérature (voir section 2.2.1) utilisent typiquement des variables spécifiant les débuts et les fins de quarts possibles ainsi que les positions des pauses. De plus, les méthodes implicites utilisent une série de contraintes liant les variables de quarts et de pauses de manière à assurer qu'à partir d'une solution optimale implicite, la solution optimale explicite peut être construite a posteriori. Ces modèles réussissent généralement à réduire significativement la taille des modèles et à résoudre très efficacement certains problèmes. De plus, quelques uns intègrent des formes de flexibilité en ce qui concerne la position des pauses. Par contre, à notre connaissance, aucun ne permet l'étude de problèmes multi-activités.

Pour mettre en relief la différence entre ces trois classes de modèle, l'exemple 1 présente un cas simple de problème de planification de quarts et un exemple de modèle pour chacune des classes.

Exemple 1 *Soit un horizon de planification de cinq périodes. Un quart peut avoir une durée de trois périodes ou de cinq périodes commençant à une période lui permettant de se terminer à l'intérieur de l'horizon de planification. Aucune pause n'est affectée à un quart de longueur 3 et une pause de 1 période doit être affectée à un quart de 5 périodes à n'importe quelle période à l'exception de la première ou de la dernière. L'objectif est de sélectionner un ensemble de quarts satisfaisant les demandes en nombre d'employés affectés à une activité de travail à chaque période de*

l'horizon de planification tel que le coût d'affectation total est minimum.

Soit $I = \{1, 2, 3, 4, 5\}$ les périodes de l'horizon de planification. Soit d_i le nombre minimum d'employés devant être affecté à une activité de travail à la période $i \in I$. Soit c_i le coût d'affectation d'un employé à une activité de travail à la période i.

Voici trois modèles pour ce problème.

Modèle de recouvrement

Soit l'ensemble Ω des 6 quarts s réalisables pour le problème :

i/s	1	2	3	4	5	6
1 :	1	0	0	1	1	1
2 :	1	1	0	0	1	1
3 :	1	1	1	1	0	1
4 :	0	1	1	1	1	0
5 :	0	0	1	1	1	1

où un 1 représente une période de travail et 0, une période de repos ou de pause. Dans le modèle de recouvrement décrit par les contraintes (2.1)-(2.3), il existe une variable x_s pour chacun des 6 quarts décrit ci-haut, les coefficients a_{is} viennent directement du tableau précédent et le coût associé à un quart s est $c_s = \sum_{i \in I} a_{is} c_i$.

Modèle en nombres entiers avec variables d'affectation

Soit E un ensemble d'employés disponibles. Définissons les variables suivantes :

$$w_e = \begin{cases} 1, & \textit{si l'employé } e \in E \textit{ travaille,} \\ 0, & \textit{sinon.} \end{cases}$$

$$u_e = \begin{cases} 1, & \textit{si l'employé } e \in E \textit{ travaille sur un quart de 3 périodes,} \\ 0, & \textit{sinon.} \end{cases}$$

$$v_e = \begin{cases} 1, & \textit{si l'employé } e \in E \textit{ travaille sur un quart de 5 périodes,} \\ 0, & \textit{sinon.} \end{cases}$$

$$x_{ei} = \begin{cases} 1, & \textit{si l'employé } e \in E \textit{ est affecté à une activité de travail à la période } i, \\ 0, & \textit{sinon.} \end{cases}$$

$$y_{ei} = \begin{cases} 1, & \textit{si l'employé } e \in E \textit{ est en pause à la période } i, \\ 0, & \textit{sinon.} \end{cases}$$

Le modèle suivant représente un modèle de type nombres entiers avec variables d'affectation.

$$Min \qquad \sum_{e\in E}\sum_{i\in I} c_i x_{ei} \tag{2.4}$$

$$\sum_{e\in E} x_{ei} \geq d_i \qquad \forall\ i \in I, \tag{2.5}$$

$$w_e = u_e + v_e \qquad \forall\ e \in E, \tag{2.6}$$

$$\sum_{i\in I} x_{ei} = 3u_e + 4v_e \qquad \forall\ e \in E, \tag{2.7}$$

$$\sum_{i\in I} y_{ei} = v_e \qquad \forall\ e \in E, \tag{2.8}$$

$$y_{e1}, y_{e5} = 0 \qquad \forall\ e \in E, \tag{2.9}$$

$$w_e, u_e, v_e \in \{0,1\} \qquad \forall\ e \in E, \tag{2.10}$$

$$x_{ei}, y_{ei} \in \{0,1\} \qquad \forall\ e \in E, i \in I. \tag{2.11}$$

Les contraintes (2.5) assurent que le nombre d'employés requis est satisfait. Les contraintes (2.6) spécifient que si un employé travaille, il doit travailler sur un quart de 3 périodes ou un quart de 5 périodes (4 périodes de travail). Les contraintes (2.7) assurent que le nombre total de périodes travaillées est correct. Les contraintes (2.8) et (2.9) restreignent les employés qui sont affectés à un quart de 5 périodes à avoir exactement une pause à l'intérieur du quart. Notons que ce modèle n'est pas unique.

Modèle implicite

Soit l'ensemble Θ des 4 types de quarts s réalisables pour le problème :

i/s	1	2	3	4
1 :	1	0	0	1
2 :	1	1	0	1
3 :	1	1	1	1
4 :	0	1	1	1
5 :	0	0	1	1

Pour le type de quart 4*, voici l'ensemble* Δ_4 *des 3 pauses p possibles :*

i/p	1	2	3
1 :	0	0	0
2 :	1	0	0
3 :	0	1	0
4 :	0	0	1
5 :	0	0	0

Pour chaque quart s, il existe une variable x_s *déterminant le nombre d'employés affectés au type de quart s et les coefficients* a_{is} *sont décrits dans le premier tableau. Pour chaque pause p du quart* $s = 4$*, il existe une variable* y_{4p} *déterminant le nombre d'employés affectés au quart* 4 *et à la pause p. Les coefficients* b_{ip} *sont donnés dans le deuxième tableau.*

Le modèle suivant représente un modèle de type implicite :

$$Min \qquad \sum_{s\in\Theta}(\sum_{i\in I} a_{is}c_i)x_s - \sum_{p\in\Delta_4}(\sum_{i\in I} b_{ip}c_i)y_{4p} \tag{2.12}$$

$$\sum_{s\in\Theta} a_{is}x_s - \sum_{p\in\Delta_4} b_{ip}y_{4p} \geq d_i \qquad \forall\ i \in I, \tag{2.13}$$

$$x_4 = \sum_{p\in\Delta_4} y_{4p}, \tag{2.14}$$

$$x_s \geq 0 \ \textit{et entier} \qquad \forall\ s \in \Theta, \tag{2.15}$$

$$y_{4p} \geq 0 \ \textit{et entier} \qquad \forall\ p \in \Delta_4. \tag{2.16}$$

Les contraintes (2.13) assurent que le nombre d'employés qui sont affectés à un type de quart et qui ne sont pas en pause satisfait la demande à chaque période. La contrainte (2.14) permet d'assurer que tous les employés affectés au type de quart 4 recevront une pause réalisable à l'intérieur de ce type de quart. Cette contrainte assure qu'à partir d'une solution optimale au modèle implicite, une solution optimale explicite peut être construite. Ce modèle est similaire à l'idée du modèle de Aykin (1996) (voir section 2.2.1).

Les sections suivantes présentent les principaux travaux de recherche qui ont étudié les problèmes introduits à la section 1.

2.2.1 Problèmes de planification de quarts mono-activité

Les problèmes de planification de quarts mono-activité ont reçu beaucoup plus d'attention que les problèmes multi-activités. Dans la littérature, ce sont généralement des problèmes où les employés sont anonymes. Nous allons répertorier dans cette sous-section seulement les principales contributions. Nous nous concentrerons sur les différentes formulations, les méthodes de résolution exactes et les modèles comportant toutes les contraintes du problème, laissant les méthodes heuristiques et les méthodes décomposant le problème en plusieurs étapes de côté, à moins qu'elles ne présentent des modèles dont la structure est pertinente relativement aux modèles présentés dans cette thèse.

Approches explicites

Le modèle de Dantzig (1954) décrit dans la section précédente représente chaque quart de manière explicite. Cette modélisation permet une flexibilité de modélisation très grande, aussi bien dans la composition des quarts de travail que dans la représentation des coûts associés à ces quarts. L'idée de base du modèle de Dantzig (1954) peut donc facilement être appliquée à d'autres contextes comme les cas multi-activités et les cas personnalisés de problèmes de planification de quarts.

Segal (1974) étudie un problème de planification de quarts avec une liste de 30 quarts prédéterminés par des heures de début et de fin fixées et les périodes de pause idéales devant leur être attribuées. Le problème est formulé comme un problème de recouvrement, mais est résolu en deux étapes. La première étape considère le problème de planification des quarts sans considérer les pauses. Segal utilise un modèle de flot dans les réseaux pour identifier les quarts optimaux. La seconde étape affecte de manière heuristique les pauses aux quarts en utilisant un autre réseau et un problème de flot à coût minimum.

Morris et Showalter (1983) présentent un problème de planification de quarts où tous les quarts sont identiques : 4 heures de travail, 1 heure de pause et 4 heures

de travail supplémentaires. L'horizon de planification de 24 heures étant divisée en périodes de 1 heure, le modèle de recouvrement résultant est de très petite taille. Pour le résoudre, ils proposent un algorithme de séparation et évaluation progressive avec l'utilisation de plans coupants sur la valeur de la borne inférieure du nombre de quarts nécessaires, qu'ils souhaitent minimiser. La méthode est générale et pourrait en théorie s'appliquer sur des problèmes de plus grande taille.

Mehrotra *et al.* (2000) proposent une extension du modèle de recouvrement de Dantzig (1954) qui tient compte des bornes supérieures et inférieures sur le nombre de quarts pouvant être affectés à une période de travail et sur le nombre de quarts étant affectés à une période de pause à chaque période. Cette extension permet de prendre en considération, entre autres, les fenêtres de temps sur le moment dans un quart où peut être pris une pause. Mehrotra *et al.* (2000) résolvent le problème de recouvrement par une méthode de génération de colonnes avec des stratégies de branchement adaptées au problème. Les colonnes sont générées par une méthode d'énumération complète et considérées comme une unique suite de séquences de travail et de pauses. Cette façon de les représenter fait qu'en tenant compte des variables duales du problème maître, une colonne donnée est facilement identifiable. Les branchements sur les périodes de travail, de pauses, de séquences de travail et autres branchements spécifiques sont tenus en compte dans le sous-problème en gardant les valeurs nécessaires dans des structures adaptées. Les résultats montrent que la méthode est très compétitive en comparaison avec le modèle de Aykin (1996) sur un grand nombre d'instances.

Approches implicites

Pour pallier la croissance rapide du nombre de variables des modèles de type explicite lorsque le nombre de quarts devient très élevé, plusieurs approches implicites ont été suggérées pour le cas anonyme. Dans cette section, nous observons différentes formulations implicites sans porter une attention particulière aux méthodes de résolution. La plupart de ces modèles pourraient être résolus par un logiciel commercial de résolution de modèle linéaire en nombres entiers, comme les modèles suggérés dans cette thèse, à l'exception du modèle de génération de colonnes que nous considérons dans une autre classe de modèles et méthodes.

Moondra (1976) développe un modèle linéaire pour le problème de planification de quarts de travail pour un environnement contenant des employés à temps partiel et à temps plein. Les quarts des employés à temps partiel sont représentés implicitement avec une variable pour chaque heure de début possible et chaque heure de fin possible représentant le nombre d'employés commençant et finissant à ces heures. Des contraintes assurent que les quarts générés ont des longueurs permises.

Gaballa et Pearce (1979) étudient le problème de planification de quarts de travail dans un environnement où 7 quarts existent et où les pauses de dîner doivent être placées à l'intérieur d'une fenêtre de temps donnée. Le modèle contient des variables pour le nombre d'employés affectés à chacun des quarts et des variables déterminant le nombre d'employés affectés à chaque pause permise pour chaque type de quarts. Des contraintes assurent que pour chaque quart, un nombre suffisant de pauses est disponible dans la solution.

Bechtold et Jacobs (1990) proposent aussi un modèle implicite pour la planification de quarts de travail dans un environnement pouvant contenir plusieurs types de quarts, chacun associé à une fenêtre de temps dans laquelle doit être affectée une pause. Une pause n'est plus directement reliée à un quart, mais définie simplement par son heure de début et l'ensemble des pauses possibles est déterminé à partir des fenêtres de temps de tous les types de quarts. Des variables sont définies pour chaque type de quarts et chaque pause. Des contraintes assurent que les nombres de types de quarts sont cohérents avec les nombres de pauses, c'est-à-dire que dans la solution chaque quart pourra être associé à une pause positionnée dans sa fenêtre de temps. Ces contraintes sont de deux types, les contraintes *forward* et les contraintes *backward* et assurent que pour un intervalle de temps, le nombre de pauses affectées est supérieur ou égal au nombre de quarts pouvant être associé à cette pause. Le modèle diminue significativement le nombre de variables requis pour le même problème avec un modèle de recouvrement.

Pour que leur approche soit valide, Bechtold et Jacobs (1990) supposent qu'il n'existe pas d'extra-chevauchement, c'est-à-dire qu'en aucun cas une fenêtre de temps associée à un quart est strictement incluse dans la fenêtre de temps d'un autre quart. Addou et Soumis (2007) proposent une façon de pallier cette restriction en ajoutant un petit ensemble de contraintes.

Thompson (1995a) propose une formulation qui représente implicitement les quarts en s'inspirant de l'idée de Moondra (1976) et les pauses (une pause par quart) de manière similaire à Bechtold et Jacobs (1990). Il utilise différents ensembles de contraintes pour lier les débuts de quarts aux fins de quarts, pour assurer que les longueurs des quarts sont correctes, pour restreindre les périodes de temps de travail avant et après la pause et finalement, il utilise des contraintes pour assurer que chaque quart a une pause qui peut lui être associée.

Aykin (1996) développe une formulation implicite qui peut représenter implicitement plusieurs catégories de pauses (par exemple, deux pauses café et une pause dîner), chacune associée à une fenêtre de temps, dans chaque type de quart. La formulation est basée sur l'idée de Gaballa et Pearce (1979). La formulation de Aykin (1996) contient des variables de pauses associées à chaque type de quarts et chaque catégorie de pauses du quart ainsi que des ensembles de contraintes assurant que chaque quart a le bon nombre de pauses pour chaque catégorie de pauses.

Rekik *et al.* (2004) présentent une formulation pour le problème de planification de quarts basée sur un problème de transport qui inclut des variables d'affectation des pauses vers les quarts ainsi qu'une formulation basée sur les contraintes *forward* et *backward* introduites par Bechtold et Jacobs (1990) (qu'ils étendent ensuite au problème de planification simultanée de cycles et de quarts de travail). La première formulation est proche de celle de Aykin (1996), mais comptent plus de variables. Rekik *et al.* (2004) utilisent cette formulation pour montrer que leur formulation basée sur un problème de transport, celle de Bechtold et Jacobs (1990) et celle de Aykin (1996) possèdent la même relaxation linéaire sous la condition que le problème ne contient pas d'extra-chevauchement. Rekik *et al.* (2004) soutiennent que les formulations de Bechtold et Jacobs (1990) et de Aykin (1996) peuvent être tour à tour plus faciles à résoudre, dépendant du contexte.

Rekik *et al.* (2010) étudient deux nouvelles formes de flexibilité dans la construction de quarts de travail implicites : la possibilité de décomposer des pauses et les contraintes de restriction sur la longueur des séquences de travail non interrompues. Deux formulations sont proposées, l'une est une extension du modèle de Bechtold et

Jacobs (1990) et l'autre, une extension du modèle de Aykin (1996). Les contraintes de restriction sur la longueur des périodes de travail sont formulées à l'aide de contraintes de type *forward* et *backward*. Le concept de pause décomposable est introduit par l'énumération des patrons de décomposition de pauses, spécifiant les longueurs des pauses décomposées et les fenêtres de temps associées.

À notre connaissance aucun modèle implicite n'est capable de traiter les problèmes de planification de quarts multi-activités.

2.2.2 Problèmes de planification d'horaires de personnel multi-activités

Dans cette section nous étudions les travaux traitant des problèmes de planification de quarts multi-activités. Contrairement au cas mono-activité, dû au peu de littérature sur le sujet, nous tentons dans cette sous-section de regrouper une plus grande variété de travaux s'approchant de la définition que nous en avons faite.

Ritzman *et al.* (1976) présentent un problème de planification de cycles de travail où les différentes opérations devant être effectuées pendant le cycle requièrent certaines qualifications de la part des employés qui sont de deux types : temps plein et temps partiel. Les auteurs soutiennent que le problème peut être formulé avec des variables d'affectation binaires spécifiant si un opérateur est affecté ou non à un cycle effectuant une opération donnée à un moment donné du cycle, mais que la complexité et la taille du problème rend la résolution directe d'un tel modèle impossible en pratique. Ils suggèrent donc une approche purement heuristique pour aborder le problème.

Loucks et Jacobs (1991) abordent un problème qui ressemble de plus près au problème étudié dans cette thèse. Un horizon de planification d'une semaine est divisé en périodes d'une heure. Pour chaque période est donné le minimum d'employés requis pour faire différentes activités de travail (8 activités dans le cas étudié). Les employés ont différentes qualifications et périodes de disponibilité. Des contraintes sur le nombre d'heures total et jours de travail pouvant être affecté à un employé dans la semaine sont présentes dans le problème. Le positionnement des pauses et des contraintes sur

le nombre d'heures devant séparer deux quarts consécutifs ne sont pas pris en compte. Loucks et Jacobs (1991) proposent une formulation avec des variables d'affectation binaires et, tout comme Ritzman *et al.* (1976), ils soutiennent que le problème ne peut être résolu directement et proposent une heuristique.

Böehmer et Grüener (2003) proposent une reformulation du modèle présenté par Loucks et Jacobs (1991) pour pallier les faiblesses qu'ils identifient dans cette dernière. En effet, Böehmer et Grüener (2003) présentent deux contre-exemples montrant que le modèle de Loucks et Jacobs (1991) permet des solutions où les employés peuvent travailler des heures non-consécutives dans une même journée, alors que la description du problème l'empêche. Böehmer et Grüener (2003) soutiennent que leur formulation décrit plus exactement le problème défini par Loucks et Jacobs (1991) puisqu'ils éliminent ces cas en remplaçant certaines contraintes du modèle initial. Par contre, ils ne présentent pas de résultats expérimentaux.

Omari (2002), Vatri (2001) et Bouchard (2004) étudient différents aspects du problème d'affectation d'activités de travail à l'intérieur de quarts de travail. Omari (2002) étudie le problème d'affecter des activités à des quarts prédéterminés, Vatri (2001) considère simultanément la génération de quarts et l'affectation des activités et Bouchard (2004) étudie en plus le placement des pauses. Les trois approches se basent sur des formulations de réseaux multi-flots avec ressources et contraintes supplémentaires. Leurs méthodes de résolution sont basées sur des méthodes heuristiques de séparation et d'évaluation progressive et de génération de colonnes (avec sous-problèmes de plus courts chemins avec contraintes de ressources) intégrées à une approche d'horizon roulant. La façon dont ils abordent les problèmes découpés et les réseaux sous-jacents diffèrent pour chacun des travaux.

Lequy *et al.* (2009) traitent un cas simplifié du problème introduit par Omari (2002), où les quarts (et les pauses) sont construits et affectés a priori aux employés. Le problème est d'affecter des activités aux quarts en prenant en compte les qualifications des employés tout en minimisant les coûts de sous-couvertures et sur-couvertures ainsi que les coûts associés aux changements d'activité. Trois modèles mathématiques sont suggérés. Le premier est un réseau multi-flots avec contraintes supplémentaires. Le deuxième est une reformulation du premier qui réduit le nombre de contraintes et de variables. Le troisième est un modèle de génération de colonnes

dérivé du modèle de multi-flots. Lequy *et al.* (2009) développent une méthode de séparation et d'évaluation progressive tronquée ainsi qu'une heuristique qui fixe à zéro toutes les variables ayant une coût réduit strictement positif dans la relaxation linéaire pour accélérer la recherche de solution entière pour le deuxième modèle. Pour trouver des solutions entières avec le modèle de génération de colonnes, deux méthodes heuristiques sont suggérées. Premièrement, une solution entière est recherchée directement avec les colonnes générées à la racine de l'arbre et après un temps limite, si nécessaire, la seconde heuristique est lancée. Celle-ci arrondit à 1 une variable entière dont la valeur est près de 1, puis résout à nouveau la relaxation linéaire à l'aide de la génération de colonnes. Ces deux étapes sont recommencées jusqu'à ce qu'une solution entière soit obtenue. Finalement, le modèle de génération de colonnes et ses heuristiques, et le deuxième modèle en nombres entiers et ses deux heuristiques de recherche de solutions entières sont intégrés dans un horizon roulant. De bons résultats expérimentaux sont notamment obtenus avec la méthode de génération de colonnes heuristique intégrée dans l'horizon roulant. Dans la présente thèse, nous aborderons aussi ce problème avec une approche différente.

Demassey *et al.* (2006) présentent une méthode de génération de colonnes basée sur un sous-problème de programmation par contraintes abordant des problèmes de planification de quarts multi-activités. Cette méthode et d'autres méthodes de programmation par contraintes sont décrites à la section 2.3.4.

2.2.3 Approches basées sur des réseaux pour les problèmes de planification d'horaires de personnel

Puisqu'une partie du travail de cette thèse comporte une formulation en réseau, nous présentons ici quelques approches contenues dans la littérature de planification d'horaires de personnels (sans se restreindre aux problèmes de planification de quarts) se basant aussi sur des réseaux. La grande majorité des modèles répertoriés dans cette section abordent le problème de planification de cycles de travail, souvent appelé problème de planification de tours. Ce problème vise à identifier sur un horizon de planification de plusieurs jours, quel type de quart (ex. quart de jour, de soir ou de nuit) est affecté à chaque jour et où sont placés les jours de congé.

Segal (1974), cité ci-haut, utilise un réseau comportant un noeud pour chaque début de période de l'horizon de planification et des arcs *forward* entre chaque noeud de début de période et celui de la période suivante. Les arcs *forward* ont une capacité minimum égale à la demande sur la période associée. Il introduit aussi des arcs *backward* reliant un noeud à la fin d'un quart (fixée a priori par la définition du problème) et le noeud associé à la période de début de ce même quart. Les arcs *backward* ont une capacité minimum égale au nombre d'employés minimum devant être affecté à ce quart, une capacité maximum égale au nombre d'employés disponible pour effectuer ce quart ainsi qu'un coût unitaire associé au quart. Le problème à résoudre est un problème de flot à coût minimum avec contraintes de capacité.

Balakrishnan et Wong (1990) utilisent une formulation en réseaux pour aborder le problème de planification de cycles de travail et de jours de congé. Le problème est d'identifier des cycles de travail sur un horizon de planification de quelques semaines. À chaque jour d'un cycle de travail est affecté un type de quart (par exemple, quart de jour, quart de soir, quart de nuit) ou un congé. Certaines règles régissent les séquences de types de quart, les moments où doivent être affectés les congés et un nombre d'employés requis à chaque jour pour chaque quart est donné. Balakrishnan et Wong (1990) proposent un réseau contenant toutes les contraintes de leur problème à l'exception des contraintes de demande sur le nombre d'employés requis. Le réseau compte, pour chaque jour de l'horizon de planification, un noeud par type de quart et un noeud associé à un congé. Un arc d'un noeud de congé au jour i à un noeud associé à un quart donné au jour $i + k$ signifie une séquence de k jours de travail affectés au quart. Une séquence de travail devant être suivie par une séquence de repos, tous les noeuds de quart ayant un arc entrant doivent avoir comme arc sortant un arc vers un noeud de congé. Seulement les arcs représentant des séquences de longueurs permises sont ajoutés au réseau. Les contraintes de demande en employés pour chaque jour sont dualisées et une borne inférieure est obtenue par une approche de relaxation lagrangienne. Finalement, une solution primale optimale est obtenue en énumérant les K plus courts chemins.

Le problème de planification de cycles de travail comme il est décrit par Balakrishnan et Wong (1990) peut se comparer à un problème de planification de quarts multi-activités anonymes, où les périodes sont des jours, les activités sont des quarts et les pauses sont des jours de congé.

Millar et Kiragu (1998) proposent aussi une formulation en réseau pour le problème de planification de cycles de travail où deux types de quarts (jour ou nuit) et des congés peuvent être affectés à chaque jour du cycle. Le réseau est construit à partir de séquences de travail et de congé permises prédéterminées (appelées *stints*). Les noeuds du réseau sont associés à des *stints* et des périodes (jours) de l'horizon de planification et sont reliés par des arcs s'il est possible de passer d'une *stint* à une autre dans le problème donné. Par exemple, la séquence *jour-jour-nuit-nuit* pourrait être suivie par la séquence *congé-congé*, mais pas par la séquence *jour-jour*. Le problème est défini comme un problème de flot à coût minimum avec des contraintes additionnelles pour capturer toutes les restrictions du problème.

Çezik *et al.* (1999) présentent une formulation qui intègre à la fois un problème de planification de cycles de travail sur 7 jours à l'aide d'une formulation en réseaux et 7 problèmes de planifications de quarts mono-activité anonymes. Les problèmes de planification de quarts journaliers sont formulés comme un problème de recouvrement et peuvent être résolus par une méthode implicite adaptée, selon le degré de flexibilité nécessaire. Le réseau décrit pour le problème de cycle de travail est un réseau en couches de temps qui peut prendre en compte le nombre de jours de congé, des contraintes sur les variations d'heure de début des quarts entre une journée et la suivante en combinant des quarts générés par les problèmes de planification de quarts journaliers. La formulation résultante est résolue par une heuristique de fixation de variables et une méthode de séparation et évaluation progressive.

Sodhi (2003) décompose un problème de planification de cycles de travail sur plusieurs semaines en deux étapes : la génération du cycle complet qui spécifie si un jour donné est affecté à un repos ou à un des trois quarts possibles (jour-soir-nuit) et l'affectation d'une tâche spécifique aux quarts avec une heure plus précise de début et de fin. La première étape est formulée par un graphe cyclique construit manuellement. Les noeuds représentent des séquences permises de combinaisons de quarts et repos sur une ou plusieurs semaines et les arcs représentent des transitions permises entre les séquences associées aux noeuds. La plupart des contraintes sont introduites dans le graphe. Des contraintes relaxées sont introduites dans l'objectif et comme contraintes supplémentaires dans un modèle linéaire en nombres entiers résolu avec

le solveur commercial CPLEX qui cherche à identifier un chemin de coût minimum dans le graphe.

Knighton et Cochran (2005) présentent une méthode en deux phases pour résoudre le problème des cycles de travail personnalisés qui sélectionne en premier lieu un ensemble de quarts et le nombre d'employés devant leur être affectés de manière à satisfaire la demande décomposée en périodes d'une heure sur un horizon de 7 jours. Tous les quarts d'une longueur de 3h à 6h peuvent être générés s'ils peuvent se terminer à l'intérieur de l'horizon de planification. Aucune pause n'est spécifiée pour les quarts. Cette étape est résolue par un problème de recouvrement. La deuxième étape consiste à affecter les quarts générés aux employés. Pour ce faire, Knighton et Cochran (2005) proposent une formulation de flots dans un réseau contenant différents types de noeuds. Premièrement, un flot égal à la quantité d'employés devant être affectés à chaque quart entre dans des noeuds associés à chacun de ces quarts. Ensuite, un arc relie chacun des noeuds-quarts à chaque noeud correspondant à un employé ayant les compétences et la disponibilité pour effectuer ce quart. Ensuite, des noeuds et des arcs assurent qu'à chaque employé est affecté un nombre permis de quarts. Chaque affectation a un poids selon la préférence de l'employé et de son superviseur et le problème est résolu comme un problème de flot à coût minimum en utilisant CPLEX. Les auteurs introduisent des contraintes linéaires notamment pour assurer qu'un employé ait suffisamment de temps de repos entre deux quarts et soutiennent que dans leurs expérimentations, ces contraintes n'affectent pas la propriété d'intégralité du modèle.

2.2.4 Génération de colonnes et branch-and-price

Pour pouvoir aborder les problèmes de planification de quarts de personnel qui peuvent faire intervenir un grand nombre de variables et de contraintes, une alternative est d'utiliser une méthode de génération de colonnes. Cette méthode permet de formuler le problème avec un modèle de type recouvrement et de l'aborder avec un sous-ensemble des variables (colonnes) en y ajoutant progressivement les variables pertinentes. Ces nouvelles variables sont générées à l'aide d'un sous-problème adapté. Dans cette section, nous présentons les grandes lignes de la méthode. Pour plus de détails sur la méthode de génération de colonnes, le lecteur peut se référer à Lasdon

(1970), Minoux (1989) et Lübbecke et Desrosiers (2005).

Méthode de génération de colonnes

Pour présenter la méthode de génération de colonnes, nous utiliserons la formulation de Dantzig (1954) présentée précédemment, puisqu'elle est adaptée aux problèmes d'horaires de personnel qui nous intéressent ici. Nous référerons au modèle des contraintes (2.1)-(2.3) avec les contraintes d'intégrité (2.3) relaxées comme le *problème maître* (PM). Le modèle suivant présente ce que nous nommerons par la suite le *problème maître restreint* à l'itération de génération de colonnes k (PMR_k).

$$Min \qquad \sum_{s \in \Omega_k} c_s x_s \tag{2.17}$$

$$\sum_{s \in \Omega_k} a_{is} x_s \geq d_i \qquad \forall\ i \in I, \tag{2.18}$$

$$x_s \geq 0 \qquad \forall\ s \in \Omega_k. \tag{2.19}$$

Notons que cette formulation est identique à celle du PM à l'exception de l'ensemble des variables Ω_k, qui varie d'une itération de génération de colonnes à l'autre. Cette formulation est donc de plus petite taille que celle du PM associé.

À une itération de génération de colonnes donnée k, le PMR_k associé est résolu. Aux contraintes du modèle sont associées des variables duales à partir desquelles seront calculés des coûts réduits permettant de déterminer si des variables de coût réduit négatif peuvent être introduites au problème maître restreint suivant. Si tel n'est pas le cas, la solution optimale du PMR_k est optimale pour le PM.

Plus précisément, soit π_i $i \in I$, les variables duales associées aux contraintes (2.18). Le coût réduit $\bar{c}_s$ d'une variable s est donc :

$$\bar{c}_s = c_s - \sum_{i \in I} \pi_i a_{is} \tag{2.20}$$

Supposons que nous ayons un algorithme qui peut générer une variable s telle que :

$$\bar{c}_s = \min_{p \in \Omega} \bar{c}_p \tag{2.21}$$

où Ω est l'ensemble de toutes les variables de PM. Si la variable résultante s est telle que $\bar{c}_s < 0$, cette variable peut être introduite dans PMR_{k+1} et faire diminuer la valeur de l'objectif courant si elle prend une valeur plus grande que 0. Si $\bar{c}_s \geq 0$, aucune variable de coût réduit négatif n'existe et la solution de PMR_k est optimale pour le PM.

L'algorithme qui peut générer la variable de coût réduit minimum se nomme le *sous-problème*. Cet algorithme doit typiquement être facile à résoudre parce qu'il peut possiblement être résolu à plusieurs reprises. Des problèmes de plus courts chemins, de sacs à dos ou des petits programmes linéaires sont souvent utilisés comme sous-problèmes. Dans certains problèmes, plusieurs sous-problèmes peuvent être nécessaires.

Recherche de solutions entières

La génération de colonnes permet de résoudre un programme linéaire. Dans notre cas, il s'agit du modèle de Dantzig (1954) représenté par les contraintes (2.1)-(2.3) que nous nommerons D, avec les contraintes d'intégrité relaxées. Pour trouver la solution entière optimale à partir d'une solution fractionnaire optimale pour la relaxation linéaire, une procédure de branchement (*Branch-and-price*) doit être développée en plus de la méthode de génération de colonnes. Une procédure de branchement doit éliminer la solution optimale fractionnaire courante en partitionnant l'espace des solutions en deux sous-problèmes, sans éliminer de solutions entières. De plus, la procédure de branchement doit pouvoir être gérée par le sous-problème de génération de colonnes, puisqu'à chaque noeud de l'arbre de branchement, la génération de colonnes sera utilisée pour résoudre la relaxation linéaire du modèle entier courant où, possiblement, de nouvelles colonnes seront introduites aux problèmes maîtres restreints associés aux noeuds. Lübbecke et Desrosiers (2005) présentent une section sur la recherche de solutions entières dans un contexte de génération de colonnes.

2.3 Programmation par contraintes et langages formels

2.3.1 Modélisation par programmation par contraintes

Dans la présente section, nous introduirons quelques notions de programmation par contraintes nécessaires à la compréhension de la suite des travaux de la présente thèse (pour plus d'information sur le sujet, voir Apt (2003)).

La modélisation par programmation par contraintes (PPC) est basée sur un ensemble de variables X_i ayant un domaine fini D_{x_i}. Ces variables sont liées par un ensemble de contraintes mathématiques ou symboliques. Une contrainte impose des restrictions sur les combinaisons de valeurs que peuvent prendre les variables. Le moteur de résolution à la base de la PPC est fondé sur la propagation de contraintes qui diffuse à travers l'ensemble des contraintes des résultats de raisonnement locaux obtenus par des algorithmes de filtrage. Chaque type de contrainte possède son propre algorithme de filtrage pour éliminer des domaines des variables qui la composent des valeurs incohérentes.

Un des intérêts de la PPC vient de l'expressivité de contraintes dites globales. Ces contraintes symboliques permettent non seulement d'exprimer directement certaines restrictions à modéliser, mais elles sont aussi associées à des algorithmes de propagation adaptés qui permettent la mise à jour dynamique (le filtrage) des domaines des variables qui les composent. La section 2.3.2 présente deux contraintes globales basées sur des langages formels et les structures utilisées pour assurer la cohérence des domaines des variables qui les composent.

2.3.2 Langages formels appliqués à la programmation par contraintes

Un langage est un ensemble de combinaisons de symboles (ou de lettres) d'un alphabet, appelées mots. La fonction associant l'alphabet au langage peut être un automate ou une grammaire. Ces derniers permettent de déterminer si un mot fait partie d'un langage ou non. Dans ce qui suit, nous allons définir les bases de ce que sont les automates et les grammaires dites hors-contextes et la façon dont ils sont utilisés pour définir des contraintes globales en programmation par contraintes. Pour

plus d'information sur la théorie des langages, nous suggérons la référence suivante : Hopcroft *et al.* (2001).

Automates

Un automate fini déterministe (AFD) est décrit par 5 éléments $\Pi = (Q, \Sigma, \delta, q_0, F)$ où :

- Q est un ensemble fini d'états ;
- Σ est un alphabet ;
- $\delta : Q \times \Sigma \rightarrow Q$ est une fonction de transition ;
- $q_0 \in Q$ est l'état initial ;
- $F \subseteq Q$ est un ensemble d'états finaux.

Les langages réguliers sont les langages reconnus par un AFD. Les automates non-déterministes (AFN) engendrent les mêmes langages que les automates déterministes, mais se distinguent de ceux-ci par le fait que leurs transitions ne définissent pas nécessairement une fonction, i.e. plus d'une transition peut quitter un état vers des états différents avec le même symbole. Certains langages peuvent être encodés par un AFN comportant moins d'états et de transitions qu'un AFD. Dans ces cas, pour l'usage qui en est fait dans les travaux de cette thèse, l'utilisation de l'AFN devrait être favorisée. Dans la plupart des cas qui nous concernent, le non-déterminisme n'est pas nécessaire.

Pour qu'un mot fasse partie d'un langage défini par un automate, celui-ci doit être reconnu par l'automate. Pour le vérifier, le mot est traité de la manière suivante. Le traitement commence en se plaçant à l'état initial de l'automate. Puis, les lettres du mot sont traitées une à une en suivant les transitions leur correspondant de manière à passer d'un état à l'autre. Si, à la fin du mot, l'état courant est un des états finaux, le mot est reconnu par l'automate et fait partie du langage défini par celui-ci. Sinon, le mot n'est pas reconnu par l'automate et ne fait pas partie du langage.

Exemple 2 *Soit* $\Sigma = \{a, b, c\}$*, un alphabet.* Π*, représenté à la figure 2.1, est un AFD reconnaissant un langage sur cet alphabet. À titre d'exemple, les mots abba, abaaaa et cc sont reconnus par l'automate. Par contre, le mot abbc ne l'est.*

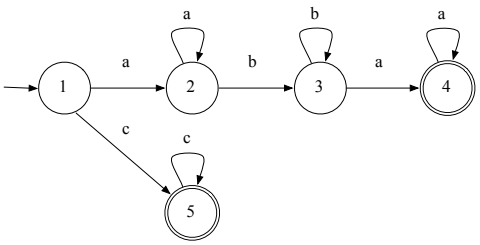

FIGURE 2.1 Automate Π où les états sont représentés par des cercles, les états finaux, par des cercles doubles et les transitions, par des arcs.

Contrainte `regular`

Pesant (2004) a introduit la contrainte de programmation par contraintes suivante, basée sur l'appartenance des valeurs prises pas une séquence de variables à un langage régulier. Elle est définie comme suit :

Définition 1 ***Contrainte d'appartenance à un langage régulier (Contrainte `regular`).***

Soit $\Pi = (Q, \Sigma, \delta, q_0, F)$ *un automate, soit* $L(\Pi)$ *le langage associé à* Π *et soit* $X = x_1, x_2, \ldots, x_n$ *une séquence finie de variables de programmation par contraintes possédant les domaines finis suivants :* $D_1, D_2, \ldots, D_n \subseteq \Sigma$*. Alors*

$$\textit{regular}(X, \Pi) = \{(d_1, \ldots, d_n) | d_i \in D_i, d_1 d_2 \ldots d_n \in L(\Pi)\}$$

L'intérêt de cette contrainte pour le reste de notre travail réside dans la modélisation du problème par un automate et dans le graphe à couches créé par la contrainte pour l'algorithme de filtrage des domaines des variables. Un *graphe en couches* est un graphe où les noeuds sont partitionnés en couches ordonnées et où les noeuds ne peuvent être reliés qu'aux noeuds des couches adjacentes.

Soit X une séquence de variables sur laquelle est appliquée la contrainte `regular`, soit n la longueur de cette séquence et k le nombre d'états de l'automate Π. Soit C le graphe à couches associés. C possède $n + 1$ couches $(N^1, N^2, \ldots, N^{n+1})$. Chaque couche compte au plus k noeuds, associés aux états de l'automate. Nous notons q_k^i le noeud représentant l'état k de Π sur la couche i de C. La procédure `initialiser`() (Pesant (2004)) crée C. À la fin de cette procédure, chaque arc sortant d'une couche i de C représente une valeur cohérente dans le domaine D_i et chaque chemin de l'état

initial q_0 à un état final q_f^{n+1}, $f \in F$ représente une affectation admissible de valeurs aux variables X.

La figure 2.2 présente le graphe à couches associé à l'automate de la figure 2.1 pour une séquence de longueur 5 et des domaines initiaux $D_1 = D_2 = \ldots = D_5 = \{a, b, c\}$.

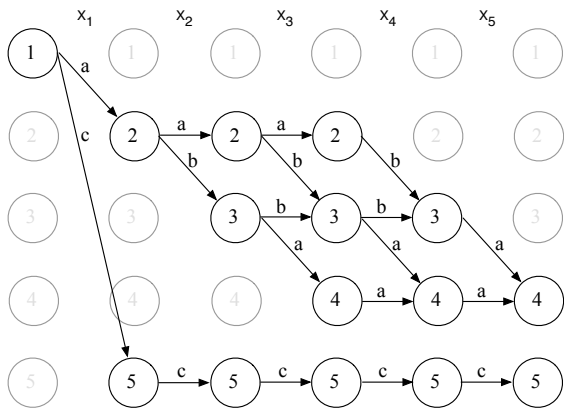

FIGURE 2.2 Graphe à couches construit par la procédure `initialiser`() pour `regular`($x_1, x_2, \ldots, x_5$, Π)

Grammaires hors-contextes

Une grammaire hors-contexte en forme normale de Chomsky est définie par 4 éléments $G = (\Sigma, N, P, S)$ où :

- Σ est un alphabet ;
- N est un ensemble fini de non-terminaux ;
- P est un ensemble de productions de la forme $A \rightarrow \gamma$ où $A \in N$ et $\gamma \in (N \times N) \cup \Sigma$;
- S est le symbole non-terminal de départ.

Un mot appartient à un langage hors-contexte si et seulement si il existe un arbre d'analyse (*parse tree*) ayant S comme racine et dont les feuilles, en ordre de gauche à droite, forment ce mot. Un arbre d'analyse pour un non-terminal est un arbre possédant ce non-terminal comme racine, des symboles de l'alphabet (aussi nommé terminaux) sur les feuilles et des non-terminaux sur les noeuds intérieurs. Dans le

cas d'une grammaire hors-contexte en forme normale de Chomsky, un non-terminal A a soit deux enfants B et C, où $A \rightarrow BC$ est une production de la grammaire, soit un enfant t où $A \rightarrow t$ est une production de la grammaire et t, un terminal. Les arbres d'analyse représentent des dérivations de mots à partir du non-terminal S par l'application successive des productions qui sont en fait des règles de réécriture du non-terminal à gauche de la production par les non-terminaux ou le terminal à droite de la production.

Exemple 3 *Considérons la grammaire G suivante :*
$G = (\Sigma = (a, b), N = (S, A, B), P, S)$, *où P est :*
$S \rightarrow AB, \qquad A \rightarrow AA \mid a, \qquad B \rightarrow BB \mid b$

*La figure 2.3 présente les deux arbres d'analyse correspondant à des mots de longueur 3 reconnus par la grammaire G. Le Tableau 2.1 représente une dérivation du mot aab à partir des productions de la grammaire. La colonne **P** représente la production utilisée et la colonne **SC** est la séquence courante, obtenue suite à l'application de la production à sa gauche.*

Contrainte `grammar`

Suite à l'introduction de la contrainte `regular`, Sellmann (2006) et Quimper et Walsh (2006) étendent simultanément l'idée d'utiliser les langages formels comme contrainte globale en restreignant les valeurs prises par une séquence de variables de programmation par contraintes à appartenir à une grammaire hors-contexte.

Tableau 2.1 Dérivation du mot *aab* à partir de la grammaire G de l'exemple 3

P	**SC**
–	S
$S \rightarrow AB$	AB
$A \rightarrow AA$	AAB
$A \rightarrow a$	aAB
$A \rightarrow a$	aaB
$B \rightarrow b$	aab

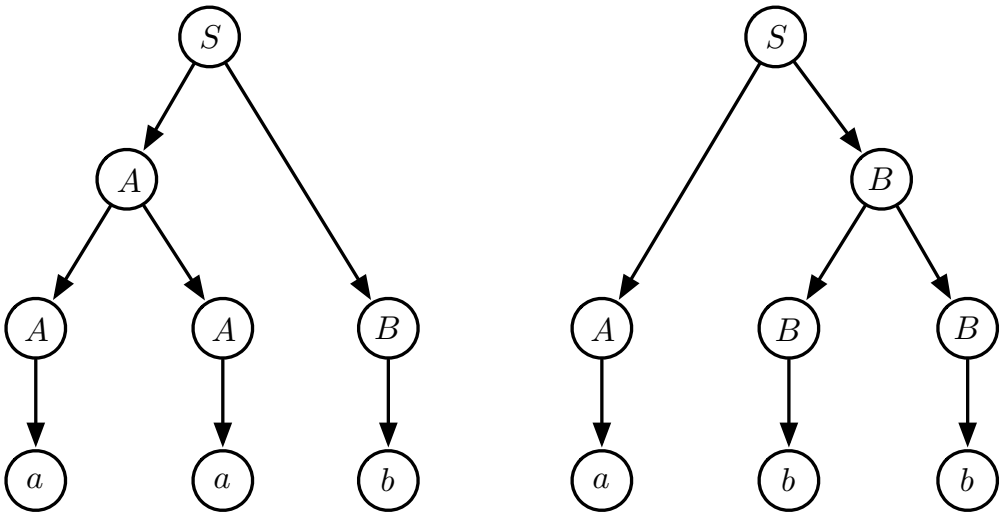

FIGURE 2.3 Arbres d'analyse pour les mots de longueur 3 reconnus par la grammaire G de l'exemple 3

Définition 2 ***Contrainte d'appartenance à un langage hors-contexte (Contrainte `grammar`).***

Soit $G = (\Sigma, N, P, S)$ une grammaire hors-contexte, soit $L(G)$ le langage associé à G et soit $X = x_1, x_2, \ldots, x_n$ une séquence finie de variables possédant les domaines finis suivants : $D_1, D_2, \ldots, D_n \subseteq \Sigma$. Alors

$$\texttt{grammar}(X, G) = \{(d_1, \ldots, d_n) | d_i \in D_i, d_1 d_2 \ldots d_n \in L(G)\}$$

Pour assurer la cohérence des domaines, l'algorithme de filtrage de la contrainte `grammar` construit un graphe orienté acyclique (GOA) Γ qui contient tous les arbres d'analyse associés à des mots de longueur n reconnus par une grammaire $G = (\Sigma, N, P, S)$. Le GOA Γ a une structure ET/OU suggérée par Quimper et Walsh (2007) en extension à la structure initialement proposée par Sellmann (2006) et Quimper et Walsh (2006). Le GOA Γ a donc deux types de noeuds, les noeuds O (noeuds OU) qui représentent des non-terminaux de N ou des lettres de l'alphabet Σ et les noeuds A (noeuds ET) qui représentent des productions de P. Chaque noeud est caractérisé par son symbole (non-terminal, lettre ou production) et une position et une longueur correspondant à la sous-séquence qu'il génère. Soit O_{il}^{π} le noeud associé au

non-terminal ou à la lettre π qui génère une sous-séquence en position i, de longueur l. Notons que si $\pi \in \Sigma$, le noeud est une feuille et l est égale à 1. Aussi, Γ a un noeud racine défini par O^S_{1n}. De la même manière, les noeuds ET sont définis comme suit : $A^{\Pi,t}_{il}$ est le $t^{\text{ième}}$ noeud représentant la production Π générant une sous-séquence de longueur l en position i. Il y a autant de noeuds $A^{\Pi,t}_{il}$ qu'il y a de façon d'utiliser Π pour générer des séquences de longueur l en position i. $A(\Pi, i, l)$ définit l'ensemble (potentiellement vide) de tous les t.

Le GOA Γ est construit de façon à ce qu'un chemin de la racine à une feuille alternera entre des noeuds de type OU et des noeuds de type ET. Voici quelques propriétés qui définissent Γ :

- $ch(O^\pi_{il})$, les enfants d'un noeud OU O^π_{il} avec $l > 1$ sont tous les noeuds ET $A^{\Pi,t}_{il}$ tels que $\Pi : \pi \rightarrow \beta$, $\beta \in (N \times N) \cup \Sigma$ et $t \in A(\Pi, i, l)$.
- $par(O^\pi_{il})$, les parents d'un noeud OU O^π_{il} où $\pi \neq S$ est un non-terminal sont des noeuds ET de la forme $A^{\Pi,t}_{jm}$ tel que $\Pi : X \rightarrow \pi Z$ ou $\Pi : X \rightarrow Y\pi$, où $j \leq i$ et $m \geq l$.
- Chaque noeud ET $A^{\Pi,t}_{il}$ où $l > 1$ tel que $\Pi : X \rightarrow YZ$ a exactement deux enfants : O^Y_{ik} et $O^Z_{i+k,l-k-1}$, où $k < l - 1$.
- Chaque noeud ET $A^{\Pi,1}_{i1}$ tel que $\Pi : X \rightarrow a$, où $a \in \Sigma$, a un unique enfant : O^a_{i1}.
- Chaque noeud ET $A^{\Pi,t}_{il}$ a un unique parent : O^π_{il} tel que $\Pi : \pi \rightarrow \beta$, $\beta \in (N \times N) \cup \Sigma$, si $l > 1$, et $\Pi : \pi \rightarrow a$, $a \in \Sigma$, si $l = 1$.

En pratique, l'algorithme à la base de la construction du GOA Γ est inspiré de l'algorithme CYK (Cocke et Schwartz (1970); Kasami (1965); Younger (1967). Voir Hopcroft *et al.* (2001)) qui fonctionne comme suit. Soit ω_{il} une sous-séquence de longueur l en position i d'un mot ω de longeur n. À partir de la grammaire $G = (\Sigma, N, P, S)$, l'algorithme CYK détermine itérativement un ensemble E_{il} de non-terminaux pouvant dériver ω_{il} pour tous $1 \leq i \leq n$ et $1 \leq l \leq n-i$, en commençant par les ensembles E_{i1} jusqu'aux ensembles E_{1n}. Si le non-terminal $S \in E_{1n}$, ω est un mot reconnu par G. Le même principe est utilisé pour construire le GOA Γ contenant tous les arbres d'analyse générant des séquences de longueur n reconnues par la grammaire G. Pendant la construction de Γ, l'algorithme de Quimper et Walsh (2007) peut prendre en compte des restrictions concernant la longueur et la position des séquences générées par des productions données.

La figure 2.4 montre le GOA Γ associé à la contrainte `grammar` sur une séquence de longueur 3 et utilisant la grammaire de l'exemple 3.

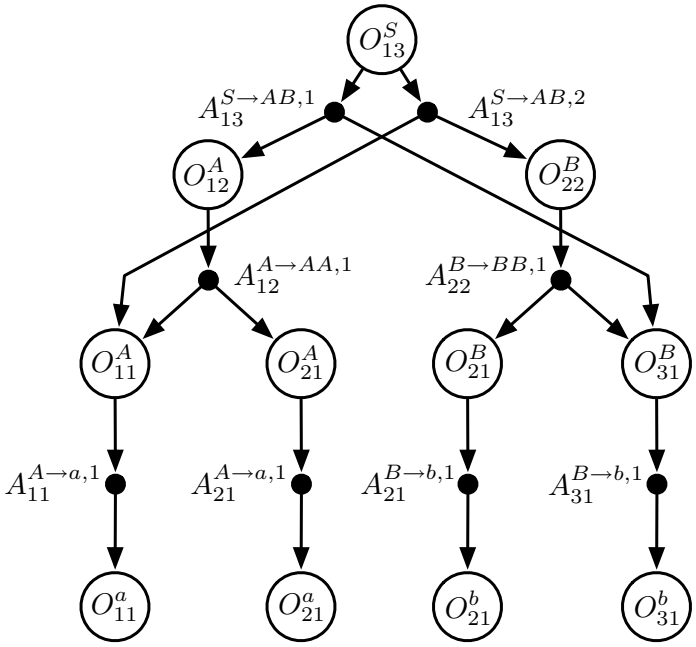

FIGURE 2.4 GOA Γ associé à la grammaire de l'exemple 3 pour des mots de longueur 3

La section suivante précise comment définir un automate et une grammaire dans le contexte des problèmes de planification de quarts de travail.

2.3.3 Définition d'automates et de grammaires pour les problèmes de planification de quarts de travail

Dans le cadre de la planification de quarts de travail, dériver un automate ou une grammaire à partir des règles régissant la composition des quarts pour un problème donné ne se fait pas de manière unique. L'objectif est de définir un langage où les mots d'une longueur égale à l'horizon de planification sont l'ensemble des quarts réalisables pour le problème. Chacune des lettres d'un mot représente donc une période de l'horizon de planification. Typiquement, l'alphabet Σ, sur lequel est basé le langage, sera composé de lettres associées à des activités de travail et de repos.

Ensuite, dans le cas d'un automate, l'état initial et les états finaux représenteront respectivement le début et la fin de l'horizon de planification. Les transitions entre ces états et les autres états de l'automate sont définies par les enchaînements permis entre les différentes activités du problème, une transition représentant une activité d'une durée égale à une période.

Pour écrire une grammaire, il faut définir, en plus de l'alphabet, des non-terminaux et des productions. Le non-terminal de départ S peut être vu comme représentant l'horizon de planification. Les autres terminaux précisent ensuite le détail des activités de la journée à partir de patrons généraux jusqu'à des patrons plus particuliers. Les productions sont, en quelque sorte, des applications de ces patrons.

L'exemple suivant présente un ensemble de contraintes sur la composition de quarts ainsi qu'un automate et une grammaire encodant ces contraintes.

Exemple 4 *Soit un problème de planification de quarts de travail contenant un ensemble d'activités de travail* $J = \{j_1, j_2, \ldots, j_n\}$. *Les règles suivantes contraignent la composition des quarts :*

- *Il existe deux types de quarts : les quarts à temps partiel et les quarts à temps plein.*
- *Les quarts à temps partiel doivent compter une pause d'une période.*
- *Les quarts à temps plein doivent compter deux pauses d'une période et un dîner de 4 périodes.*
- *Lorsqu'une activité de travail est entreprise, elle doit durer au moins 4 périodes.*
- *Une pause ou un dîner est nécessaire pour passer d'une activité de travail à une autre.*
- *Les pauses et le dîner doivent être séparés par des activités de travail.*

Définissons l'alphabet suivant : $\Sigma = \{j_1, j_2, \ldots, j_n, r, p, d\}$ *où* $j_1, j_2, \ldots, j_n$ *sont des périodes associées aux* n *activités de travail, où* r *est une période de repos, où* p *est une période de pause et* d*, une période de dîner.*

L'automate illustré à la figure 2.5 décrit l'ensemble des quarts admissibles respectant les contraintes du problème, à l'exception qu'il ne permet pas de distinguer un quart à temps plein d'un quart à temps partiel et ne permet pas de garantir le nombre de pauses prises. Ces restrictions doivent être prises en compte à l'extérieur de l'automate. L'automate est construit d'une manière telle que lorsque nous partons de l'état initial en suivant un nombre d'arcs égale à la longueur de l'horizon de planification jusqu'à l'état final, nous formons un quart qui respecte les transitions entre

les activités de travail et de repos.

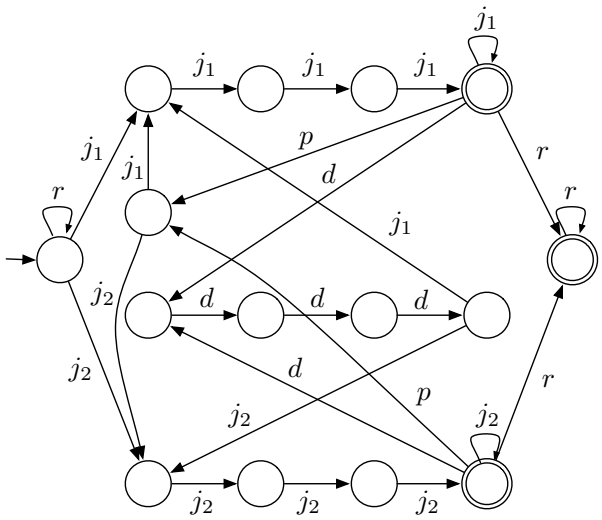

FIGURE 2.5 Automate pour le problème de planification de quarts de l'exemple 4 pour 2 activités de travail

Contrairement à l'automate, la grammaire suivante permet d'encapsuler toutes les contraintes du problème.

Définissons l'ensemble de non-terminaux N suivant :
$\{F, Q, R, T, P, D, \{J_1, J_2, \ldots, J_n\}\}$ *où F est un quart à temps plein, Q, un quart à temps partiel, R, une séquence de repos, T, une séquence de travail, P, une pause, D, une séquence de dîner et où $J_1, J_2, \ldots, J_n$, sont des séquences de travail sur les activités $j_1, j_2, \ldots, j_n$.*

Voici l'ensemble des productions associées au problème :

$S \to RFR \mid FR \mid RF \mid RQR \mid QR \mid RQ,$ $\quad P \to p,$
$F \to TPTDTPT \mid TDTPTPT \mid TPTPTDT,$ $\quad D \to dddd,$
$Q \to TPT,$ $\quad R \to Rr \mid r,$
$T \to_{[4,\infty)} J_k \quad \forall j_k \in J,$
$J_k \to J_k j_k \mid j_k \quad \forall j_k \in J.$

Notons que pour la suite des travaux présentés dans cette thèse, nous utilisons la structure du GOA Γ générée à partir d'une grammaire. Lors de la construction de cette structure, il est possible de considérer des contraintes sur les productions de manière à générer seulement des sous-arbres respectant ces contraintes dans le graphe Γ. Par exemple, la production $T \rightarrow_{[4,\infty)} J_k$ est contrainte de produire des sous-séquences de longueur plus grande ou égale à 4 périodes, donc le GOA Γ ne contiendra pas de sous-séquence de plus petite longueur générée par cette production.

Aussi, la grammaire définie dans l'exemple n'est pas en forme normale de Chomsky qui exige que les productions soient de la forme $A \rightarrow \gamma$ où $A \in N$ et $\gamma \in (N \times N) \cup \Sigma$. Pour utiliser le graphe Γ, nous supposons que la grammaire est en forme normale de Chomsky. Toute grammaire peut être transformée en forme normale de Chomsky, il ne s'agit donc pas d'une supposition restrictive. Par contre, la transformation peut être faite de différentes façons et la grammaire résultante n'est pas unique. Par exemple, la production $Q \rightarrow TPT$ peut être transformée par les productions $Q \rightarrow XT$ et $X \rightarrow TP$. Souvent, la façon dont les productions sont divisées pour obtenir la forme normale de Chomsky dépendra des contraintes que nous souhaitons mettre sur les productions. Par exemple, si on voulions contraindre l'endroit où se situe la pause dans un quart à temps partiel, la subdivision suggérée ($Q \rightarrow XT$ et $X \rightarrow TP$) serait intéressante parce qu'il serait possible de mettre une contrainte de longueur sur la production $X \rightarrow TP$ (voir les grammaires de la section 5.5.1 pour des exemples de contraintes sur les productions).

2.3.4 Langages formels, programmation par contraintes et planification de quarts de travail

Quelques approches basées sur la programmation par contraintes et les langages formels ont abordé les problèmes de planification de quarts multi-activités.

Demassey *et al.* (2006) utilisent une méthode de génération de colonnes basée sur un sous-problème de programmation par contraintes. Le sous-problème utilise la contrainte `regular` et des contraintes supplémentaires pour modéliser les contraintes sur la composition des quarts. Le problème traité dans leur travail est aussi étudié dans la présente thèse. Les instances comptent entre 1 et 10 activités de travail pour lesquelles sont données les courbes de demande sur un horizon de planification de 24

heures décomposé en périodes de 15 minutes. Le problème est d'affecter à un ensemble d'employés des quarts répondant à un ensemble de règles très flexibles : les quarts peuvent commencer à n'importe quelle période et durer entre 3h et 8h. Dépendant de la longueur du quart, des périodes de dîner et de pauses doivent être affectées au quart. La méthode suggérée est rapide pour résoudre la relaxation linéaire du problème, mais les différents branchements suggérés pour trouver des solutions entières ne réussissent pas à résoudre à l'optimalité des instances de plus de 3 activités de travail à l'intérieur d'un temps limite d'exécution de 2 heures.

Kadioglu et Sellmann (2008) proposent un algorithme de propagation pour la contrainte `grammar` fonctionnant aussi avec le GOA Γ initialement introduit pour la contrainte, mais de manière incrémentale. De cette façon, le temps et l'espace nécessaire pour mettre à jour les domaines des variables lors de la résolution sont réduits. Ils comparent les deux approches de propagation sur une version simplifiée du problème introduit par Demassey *et al.* (2006). L'algorithme incrémental améliore le temps total de résolution de 46 à 188 fois en comparaison avec la version non-incrémentale sur les problèmes ayant une et deux activités de travail.

Quimper et Rousseau (2009) présentent une approche de recherche à voisinage large pour résoudre le problème introduit par Demassey *et al.* (2006) modélisé à l'aide d'un automate ou d'une grammaire. Un ensemble aléatoire de quarts est d'abord généré. Ensuite, chaque quart est traité à tour de rôle en calculant le coût d'amélioration de l'activité affectée à chaque position par toutes les autres activités, tenant compte de l'horaire global courant, des demandes et des coûts de couverture, sous-couverture et sur-couverture du problème. Pour la modélisation utilisant la grammaire, les coûts calculés sont associés aux noeuds du GOA Γ (voir section 2.3.2) qui correspondent, chaque feuille de Γ étant définie par une position dans le quart et une activité, soit une activité de repos, soit une activité de travail. Un algorithme de programmation dynamique est ensuite utilisé pour trouver le quart qui diminue le plus le coût global. L'algorithme traite chaque noeud, des feuilles jusqu'à la racine en choisissant le noeud enfant de plus faible coût à un noeud de type OU et en sommant les enfants des noeuds de type ET. Lorsqu'un minimum local est atteint, un nouvel ensemble aléatoire de quarts est généré pour tenter de trouver une solution de coût inférieur. La méthode permet de trouver de bonnes solutions pour des instances

contenant jusqu'à 10 activités de travail.

Menana et Demassey (2009) proposent une nouvelle contrainte globale qui encapsule une contrainte `regular` avec plusieurs contraintes cumulatives spécialement pour aborder des problèmes de planification de quarts de travail. Les contraintes cumulatives permettent de borner le nombre d'occurences de certaines valeurs dans la séquence ainsi que de prendre en considération des coûts associés à cette séquence. L'algorithme de propagation est basé sur la relaxation lagrangienne d'une formulation de plus court chemin issue du graphe en couches proposé pour la propagation de la contrainte `regular` avec des contraintes supplémentaires bornant les coûts et différentes valeurs dans la séquence. Avec les règles sur la composition des quarts venant de Demassey *et al.* (2006), ils réussissent à obtenir le quart de plus petit coût (pour un employé) pour des instances allant jusqu'à 50 activités de travail en moins de 15 secondes.

Chapitre 3

DÉMARCHE DE L'ENSEMBLE DU TRAVAIL DE RECHERCHE ET ORGANISATION GÉNÉRALE DU DOCUMENT

À la lumière de la revue de la littérature concernant les problèmes de planification de quarts de travail, il ressort que certains aspects n'ont pas été étudiés en profondeur. En particulier, les méthodes génériques pour la modélisation et la résolution de problèmes de planification de quarts multi-activités sont très rares et c'est dans cette direction que notre travail de recherche s'inscrit.

La présente thèse se divise en trois parties associées à des articles. Le chapitre 4 a pour but d'introduire de nouvelles approches de modélisation qui permettent de formuler les problèmes de planification de quarts multi-activités. Les chapitres 5 et 6 présentent respectivement des méthodes efficaces pour l'étude des problèmes de planification de quarts multi-activités anonymes, puis personnalisés. Les deux approches utilisent des modèles basés sur les fondements introduits au chapitre 4. Les trois articles forment donc une suite logique établissant d'abord les bases de l'idée de modélisation pour ensuite l'adapter dans des contextes spécifiques. Nos différentes approches de modélisation et de résolution sont toutes testées sur un même cas expérimental, permettant les comparaisons. Les chapitres 5 et 6 présentent aussi d'autres cas expérimentaux de manière à se comparer plus adéquatement avec certaines méthodes existantes dans la littérature. De plus, nous présentons quelques résultats théoriques relevant les similitudes et les différences entre nos modèles et d'autres modèles reconnus.

Au chapitre 7, nous reviendrons sur ces trois chapitres afin de mettre en relief comment s'inscrit l'ensemble de notre travail de recherche en lien avec la revue de la littérature du chapitre 2. Pour finir, le chapitre 8 synthétise la thèse, présente une critique de nos travaux et introduit quelques avenues de recherche future.

Chapitre 4

FORMAL LANGUAGES FOR INTEGER PROGRAMMING MODELING OF SHIFT SCHEDULING PROBLEMS

Les règles restreignant la composition de quarts de travail comportent souvent des contraintes sur les valeurs que prennent des séquences de variables. C'est le cas de certains problèmes de planification de quarts multi-activités qui exigent qu'une pause soit prise avant de changer d'une activité à l'autre ou qui interdisent certaines activités de travail à être effectuées l'une à la suite de l'autre. Dans ces contextes et dans plusieurs autres contextes de planification de quarts de travail, il est possible d'utiliser des langages formels pour exprimer ces restrictions dans la composition des quarts.

Dans cet article, nous utilisons des automates et des grammaires hors-contextes pour définir des languages où les mots ont une longueur égale à la durée de l'horizon de planification et représentent les quarts de travail acceptables pour un problème donné. Les lettres composant ces mots sont donc les activités, de travail ou de repos, affectées à chaque période du quart en question.

En premier lieu, nous montrons comment un automate peut modéliser ces contraintes puis générer un modèle linéaire en nombres entiers 0-1 qui englobe tous les quarts réalisables pour un problème donné. À partir de l'automate, nous utilisons une structure issue d'un algorithme de programmation par contraintes pour créer une formulation de flot dans un réseau où un chemin représente un quart de travail.

En second lieu, nous utilisons aussi une structure inspirée de la programmation par contraintes pour transformer une grammaire définissant les restrictions sur la composition de quarts de travail en modèle linéaire en nombres entiers 0-1. De la gram-

maire, un graphe orienté acyclique de type ET/OU contenant tous les arbres d'analyse résultant en un quart réalisable est généré et une série de contraintes linéaires sont déduites de ce graphe. Les contraintes assurent qu'un ensemble de variables égalant 1 dans une solution forme un arbre d'analyse pour la grammaire donnée. Un arbre d'analyse étant associé à un mot, ou un quart dans le contexte qui nous intéresse, une telle solution représente une sélection de quart.

Ensuite, nous montrons que pour certains langages, les formulations dérivées des automates et des grammaires hors-contextes sont équivalentes.

Pour finir, nous étudions un cas expérimental de problème de planification de quarts de travail introduit par Demassey *et al.* (2006) pouvant faire intervenir plusieurs activités de travail. Nous comparons différentes modélisations utilisant les langages formels à une modélisation basée sur des variables d'affectation plus conventionnelle.

L'article suivant est une extension d'un article soumis à la conférence *CPAIOR 2007* (Côté *et al.* (2007)). Il a été publié en ligne le 29 octobre 2009 dans la revue *Constraints* (Côté *et al.* (2009)).

Formal Languages for Integer Programming Modeling of Shift Scheduling Problems

Marie-Claude Côté
Département de mathématiques et génie industriel, École Polytechnique de Montréal, PO Box 6079, succ. Centre-Ville, Montréal, H3C 3A7, Canada,
CIRRELT-Interuniversity Research Center on Enterprise Networks, Logistics and Transportation, Montréal, Canada, marie-claude.cote@polymtl.ca

Bernard Gendron
Département d'informatique et de recherche opérationnelle, Université de Montréal,Pavillon André-Aisenstadt, PO Box 6128, succ. Centre-Ville, Montréal, H3C 3J7, Canada,
CIRRELT-Interuniversity Research Center on Enterprise Networks, Logistics and Transportation, Montréal, Canada Bernard.Gendron@cirrelt.ca

Claude-Guy Quimper
Omega Optimisation, 4200 St-Laurent 301, Montréal, H2W 2R2, Canada,
quimper@alumni.uwaterloo.ca

Louis-Martin Rousseau
Département de mathématiques et génie industriel, École Polytechnique de Montréal, PO Box 6079, succ. Centre-Ville, Montréal, H3C 3A7, Canada,
CIRRELT-Interuniversity Research Center on Enterprise Networks, Logistics and Transportation, Montréal, Canada louis-martin.rousseau@polymtl.ca

Abstract

This paper approaches the problem of modeling optimization problems containing substructures involving constraints on sequences of decision variables. Such constraints can be very complex to express with Mixed Integer Programming (MIP). We suggest an approach inspired by global constraints used in Constraint Programming (CP) to exploit formal languages for the modeling of such substructures with MIP. More precisely, we first suggest a way to use automata, as the CP `regular` constraint does, to express allowed patterns for the values taken by the constrained sequence of variables. Secondly, we present how context-free grammars can contribute to formulate constraints on sequences of variables in a MIP model. Experimental results on both approaches show that they facilitate the modeling, but also give models easier to solve by MIP solvers compared to compact assignment MIP formulations.

4.1 Introduction

Given a sequence of n decision variables X_i, each with a finite domain D_i, $i = 1, \ldots, n$, a constraint on such a sequence is a set of n-tuples $L \subseteq D_1 \times \ldots \times D_n$ called a *language*. The constraint over the sequence is satisfied when the tuple $\langle X_1, \ldots, X_n \rangle$ belongs to the language L. Such constraints arise in many optimization and satisfaction problems. In this paper, we focus on shift scheduling problems, where a sequence of activities (work activities, break, lunch, rest) must be assigned to a set of employees. In these problems, the difficulty lies in building shifts that comply with work regulations such as legal placement of breaks and lunches, and transitions between activities.

In this paper, we study how to model constraints on sequences of decision variables using a Mixed Integer Programming (MIP) framework. Our approach is inspired by global constraints in Constraint Programming (CP) that use formal languages. First, we suggest using automata to represent constraints on sequences of decision variables, as the CP `regular` constraint (Pesant (2004)) does. From the automaton, we automatically generate a network flow model that can be included into any MIP model involving constraints on sequences of decision variables. Second, we propose a way to use context-free grammars instead of automata to describe the constraints on sequences of decision variables. To apply this to MIP, we use an and/or graph structure associated to the CP `grammar` constraint (Quimper and Walsh (2006,2007); Sellmann (2006)) and derive the associated linear constraints.

These approaches allow MIP to benefit from CP expressiveness in modeling, by automatically generating MIP models from intuitive modeling tools, such as automata and context-free grammars. Furthermore, our experimental results on a shift scheduling model show that they facilitate the modeling, but also give models easier to solve by MIP solvers compared to compact assignment MIP formulations.

The paper is organized as follows. In Section 4.2, we present a literature review on shift scheduling problems. Section 4.3 presents some background material on formal languages and their use in CP. In Sections 4.4 and 4.5, we introduce our two approaches to model constrained sequences of decision variables: the `MIP regular` and the `MIP grammar` constraints respectively. In Section 4.6, we show the equivalence of the two resulting MIP models when the grammar encodes a regular language. Finally, in Section 4.7, we present numerical results obtained by solving different

formulations of a particular shift scheduling problem.

4.2 Shift Scheduling Problems

Given a planning horizon divided into periods of equal length, a set of employees and a demand for different activities (work activities, lunch, break, rest) at each period, the shift scheduling problem consists of assigning an activity to each employee at each period in such a way that the demands are met, while optimizing an objective and satisfying several rules (including some that can be expressed as constraints on sequences of decision variables). In this context, a *shift* is a sequence of activities corresponding to a continuous presence at work (that may include lunch and break, but not rest periods). A *schedule* (also called a *tour*) is a sequence of shifts and rest periods, over the whole planning horizon, that satisfies all the rules associated with an employee. A *pattern* is a sequence of activities that respects some of the rules over a subset of the planning horizon.

Mathematical programming models for shift scheduling problems can be divided into three categories (see Ernst *et al.* (2004a,b) for recent surveys on shift scheduling and related problems): the compact assignment formulations, the explicit set covering formulations and the implicit set covering formulations. Compact assignment formulations (Balakrishnan and Wong (1990); Beaulieu *et al.* (2000); Laporte *et al.* (1980)) use decision variables to assign activities to each employee at each period. In the explicit set covering formulations (Dantzig (1954)), the decision variables represent all possible shifts and the problem is to select a subset of them which covers the demands. The number of shifts being potentially large, different methods were proposed to select good subsets. Most notably, the column generation method efficiently solves this kind of problems (see for instance Bouchard (2004); Demassey *et al.* (2006); Mehrotra *et al.* (2000)).

Implicit set covering formulations were introduced and developed by Moondra (1976), Bechtold and Jacobs (1990, 1996), Thompson (1995a), Aykin (1996, 1998), and Rekik (2006); Rekik *et al.* (2004). In these models, shift types, specified by starting and ending times, are not directly associated with break positions at first. For instance, one can independently decide how many employees are going to work from 8am to 4pm and how many employees are going to be on break at 10am. Additional constraints, named forward and backward constraints, are necessary to guarantee the

existence of a valid schedule which can later be reconstructed with a polynomial-time algorithm. The main advantage of this approach is that the number of decision variables is significantly reduced compared to explicit set covering formulations.

Network flow formulations were used for different generalizations of shift scheduling problems. Çezik *et al.* (1999) propose a MIP formulation for the Weekly Tour Scheduling Problem. It handles the weekly horizon by combining seven daily shift scheduling models in a network flow framework, which handles the demands for each day. Millar and Kiragu (1998) and Ernst *et al.* (1999) use a layered network to represent allowed transitions between a set of a priori patterns (series of nights for instance) to develop complete schedules. Sodhi (2003) studies the problem of assigning a type of shift (day, evening, night) to each day of a planning horizon of several weeks. The model combines predefined weekly patterns to create a complete schedule by using a directed graph with nodes representing allowed weekly patterns and arcs corresponding to allowed week-to-week transitions between these patterns. A MIP model is then used to find an optimal cyclic path to cover all the weeks of the schedule.

In this paper, we present two different generic ways to capture and model a large set of rules that can be expressed as constraints on sequences of decision variables. These two modeling approaches are not limited to single activity scheduling problems like implicit formulations are. Contrary to explicit and network flow formulations, they do not require a priori enumerations of shifts or patterns. Moreover, the expressiveness of automata and grammars allows to model complex rules naturally, which is not always the case with compact assignment models.

4.3 Background Material

Before we define our modeling approaches, we introduce important definitions related to formal languages theory (for more details on the subject, see Hopcroft *et al.* (2001)).

4.3.1 Automata and the CP Regular Constraint

A deterministic finite automaton (DFA) is described by a 5-tuple $\Pi = \langle Q, \Sigma, \delta, q_0, F \rangle$ where:

- Q is a finite set of states;
- Σ is an alphabet;
- $\delta : Q \times \Sigma \rightarrow Q$ is a transition function;
- $q_0 \in Q$ is the initial state;
- $F \subseteq Q$ is a set of final states.

An alphabet is a finite set of symbols. A language is a set of words, formed by symbols over a given alphabet. Regular languages are languages recognized by a DFA. A word is recognized by a DFA if by processing its symbols one by one from the initial state using the transitions, we find ourselves in a final state after we process the last symbol.

A non-deterministic finite automaton (NFA) distinguishes itself from a DFA by its set of transitions δ. In fact, a transition is no longer a function but a set of triplets : $\delta \subseteq Q \times \Sigma \times Q$. A transition $\langle q_1, j, q_2 \rangle \in \delta$ indicates that reading the symbol j from state q_1 can lead to state q_2. However, it is possible that another transition from q_1, $\langle q_1, j, q_3 \rangle \in \delta$, leads to another state q_3, hence the non-determinism of the automaton. DFAs and NFAs strictly encode the same languages. However, NFAs can encode some languages with exponentially fewer states than DFAs.

Example 1 *Let $\Sigma = \{a, b, c\}$ be an alphabet. Π, represented in Figure 4.1, is a DFA recognizing a regular language over this alphabet. This DFA recognizes, for instance, the words c, cccc, aba, aabba, but does not recognize ac and ab.*

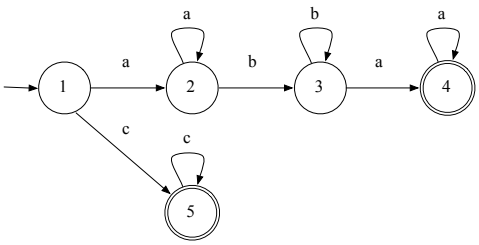

FIGURE 4.1 DFA Π with each state shown as a circle, each final state as a double circle, and each transition as an arc.

Pesant (2004) introduced the constraint `Regular`$([X_1, \ldots, X_n], \Pi)$ which is satisfied if the automaton Π recognizes the sequence of decision variables $X_1, \ldots, X_n$.

4.3.2 Context-free Grammars and the CP Grammar Constraint

A *context-free grammar* G is a tuple $\langle \Sigma, N, S, P \rangle$ where Σ is the alphabet of characters, also called the *terminal symbols*, N is a set of *non-terminal symbols*, $S \in N$ is the starting symbol, and P is a set of productions of the form $A \rightarrow w$ where $A \in N$ is a non-terminal symbol and w is a sequence of terminal and non-terminal symbols. We use capital letters for non-terminal symbols and lower case letters for terminal symbols. A *parsing tree* is a tree where each leaf is labeled with a terminal and each inner-node is labeled with a non-terminal. The root is labeled with the starting symbol S. The children of a node A, when listed from left to right, form a sequence w such that the production $A \rightarrow w$ belongs to the grammar. A grammar recognizes a sequence if and only if there exists a parsing tree where the leaves, when listed from left to right, reproduce this sequence. Any grammar can be written in its *Chomsky normal form* i.e., any production either generates two non-terminals or one terminal. A *context-free language* is the set of sequences accepted by a context-free grammar.

Context-free grammars are more expressive than automata since any regular language can be encoded with a context-free grammar but not every context-free language can be encoded with an automaton (Hopcroft *et al.* (2001)).

Quimper and Walsh (2006), and Sellmann (2006) introduced the constraint `grammar`$([X_1, \ldots, X_n], G)$ which is satisfied if the grammar G recognizes the sequence of decision variables $X_1, \ldots, X_n$.

Given a context-free grammar G, Quimper and Walsh (2007) build a Boolean formula that returns *true* for every sequence of a given length n recognized by the grammar and *false* for any other sequence. This Boolean formula is encoded in an and/or graph where each leaf corresponds to an assignment $X_i = t$ that can either be *true* or *false*. An or-node is *true* if one of its children is *true*. An and-node is *true* if all its children are *true*. The root is *true* if the grammar G accepts the sequence encoded by the leaves. Their algorithm (see Algorithm 1) builds the and/or graph and is based on the CYK parser (Cocke and Schwartz (1970); Kasami (1965); Younger (1967)) that takes as input a grammar written in its Chomsky normal form. The and/or graph embeds every possible parsing tree of a grammar. Each or-node $N(A, i, j)$ in the graph is assigned to *true* if the non-terminal A produces the sub-

sequence $X_i, \ldots, X_{i+j-1}$. The nodes set to *true* in a solution form a parsing tree. The and/or graph structure will be used in Section 4.5 to generate a MIP model from a given grammar. Example 2 describes the and/or graph that recognizes any word of length $n = 3$ defined by a simple grammar.

for *all non-terminals* A **do**
  for $i \in [1, n]$ **do**
1 Create or-node $N(A, i, 1)$
    for $A \to t \in G$ **do**
      Create the leaf node $N(t, i, 1)$ if it does not already exist.
      Create an and-node with child $N(t, i, 1)$ and parent $N(A, i, 1)$.
for $j \in [2, n]$ **do**
  for $i \in [1, n - j + 1]$ **do**
    for *all non-terminals* A **do**
      Create or-node $N(A, i, j)$
      // Create a list of children composed of and-nodes
2 $Children(N(A, i, j)) \leftarrow \{N(B, i, k) \wedge N(C, i + k, j - k) \mid$
          $k \in [1, j), \quad A \to BC \in G,$
          $Children(N(B, i, k)) \neq \emptyset,$
3 $Children(N(C, i + k, j - k)) \neq \emptyset\}$
4 Delete any node that does not have $N(S, 1, n)$ for ancestor.

Algorithm 1: This algorithm based on the CYK parser (Cocke and Schwartz (1970); Kasami (1965); Younger (1967)) constructs a graph embedding all possible parsing trees of sequences of length n recognized by the grammar G.

Example 2 *Consider the following simple grammar taken from Quimper and Walsh (2007).*

$$S \to AB \qquad A \to AA \mid a \qquad B \to BB \mid b$$

Algorithm 1 builds the graph depicted in Figure 4.2.

Quimper and Walsh (2007) show how context-free grammars can be enhanced by imposing some constraints on a production $A \to BC$. For instance, a non-literal can be constrained to produce a sequence of a given length or only be produced at given positions. Such constraints simply remove some nodes in the and/or graph.

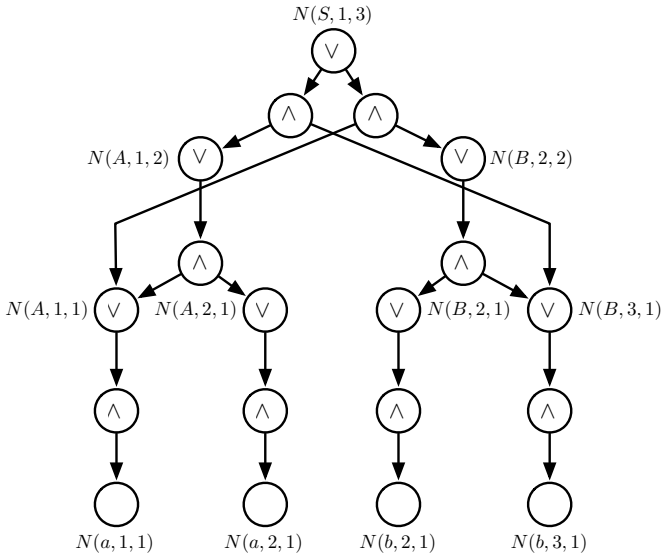

FIGURE 4.2 And/or tree constructed by Algorithm 1 on the grammar of Example 2 and a sequence of length $n = 3$.

4.4 MIP Regular

The use of automata to express constraints on values taken by sequences of variables is very useful in CP. Equivalent constraints can be very complex to formulate in a MIP model. The aim of our work in this section is precisely to propose a way to formulate MIP models by using automata. Our approach is inspired by the CP `regular` constraint (Pesant (2004)).

First, we introduce the following 0-1 decision variables:

$$x_{ij} = \begin{cases} 1, & \text{if position } i \in I \text{ of the sequence is assigned to value } j \in D_i, \\ 0, & \text{otherwise,} \end{cases}$$

where $I = \{1, 2, \ldots, n\}$ represents the set of positions in the sequence and D_i, the set of values that can be assigned to this position.

To obtain a graph structure representing all sequences of length n recognized by an automaton, we use the following property of regular languages:

– Let L_1 and L_2 be two regular languages. Then $L_1 \cap L_2$ is a regular language.

Given this property, if we have an automaton A_1 that encodes a set of constraints

on the values taken by a sequence of variables and an automaton A_2 that encodes the language specifying all sequences of length n on the same set of values, the conjunction of A_1 and A_2 results in an automaton A recognizing all sequences of length n recognized by A_1. Automaton A has a special structure. It is a directed layered graph, with $n+1$ layers and no cycles. Each layer potentially contains all states of A_1. Let $N^1, N^2, \ldots, N^{n+1}$ be the sets of states of each layer. We note that N^1 has a single element, the initial state of A_1, and that N^{n+1} is a subset of the set of final states of A_1. Pesant (2004) shows how to build A.

Since A recognizes all sequences of length n recognized by A_1, our modeling approach uses this structure to derive a network flow formulation. The correspondence between the automaton A and the graph G used for the network flow model is direct. First, a state $k \in N^i$, $1 \leq i \leq n+1$, is a node in G and a transition in A is an arc in G. A transition between a state $k \in N^i$ and a state $l \in N^{i+1}$ labeled with symbol j defines a unique arc in G representing the value j assigned to position i in the sequence. For all such arcs in G, we have a *flow variable* f_{ijkl} (see Ahuja *et al.* (1993) for details on network flow theory). Notice that if A_1 is a DFA, the index l is not needed, but our approach also applies to an NFA. Finally, we identify s, the unique element of N^1, as the *source* node, and we link each node $k \in N^{n+1}$ to a *sink* node t with an arc labeled with the flow variable $f_{(n+1)kt}$. We also define a 0-1 variable w that specifies if the constraint is active or not. The value of w corresponds to the amount of flow (0 or 1) entering and leaving the graph.

Example 3 *Let* Π *be the automaton of Example 1 represented in Figure 4.1. Let* π_5 *be the automaton depicted in Figure 4.3 representing all sequences of length* $n = 5$ *on alphabet* $\Sigma = \{a, b, c\}$*. Then, Figure 4.4 presents automaton* $A = \Pi \cap \pi_5$ *and Figure 4.5, the associated graph* G*.*

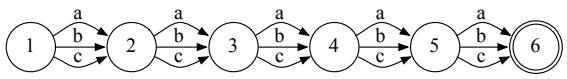

FIGURE 4.3 Automaton π_5 recognizing all sequences of length 5 on alphabet $\Sigma = \{a, b, c\}$

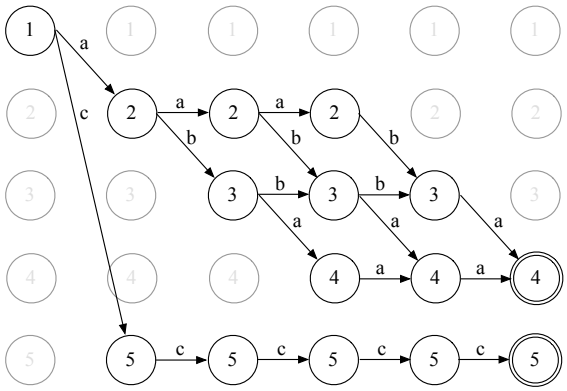

FIGURE 4.4 Automaton $A = \Pi \cap \pi_5$

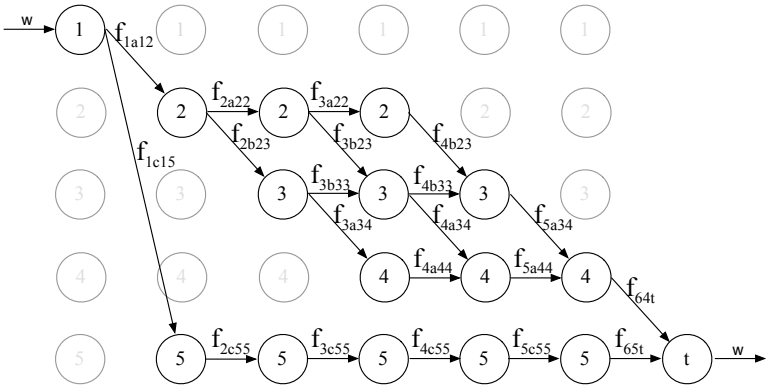

FIGURE 4.5 Graph G associated to automaton A

The network flow problem on G is a set of linear constraints, the flow conservation equations, ensuring that for each node in the graph, the amount of flow entering and leaving the node is the same. An arc from $k \in N^i$ to $l \in N^{i+1}$ with label j is defined as a quadruplet (i, j, k, l). For each node $k \in N^i$, we introduce the sets of outgoing and incoming arcs:

$$\Delta_{ik}^{+} = \{(i, j, k, l) | l \in N^{i+1} \text{ and } < k, j, l > \in \delta_i\},$$

$$\Delta_{ik}^{-} = \{(i-1, j, l, k) | l \in N^{i-1} \text{ and } < l, j, k > \in \delta_{i-1}\},$$

where δ_i is the set of transitions at layer i, $1 \leq i \leq n+1$. The MIP formulation of the `regular` constraint is then written as follows:

$$\sum_{(j,l)|(1,j,s,l)\in\Delta_{1s}^{+}} f_{1jsl} = w, \tag{4.1}$$

$$\sum_{(j,l)|(i-1,j,l,k)\in\Delta_{ik}^{-}} f_{(i-1)jlk} = \sum_{(j,l)|(i,j,k,l)\in\Delta_{ik}^{+}} f_{ijkl}, \quad \forall i \in \{2, \ldots, n\}, k \in N^i, \tag{4.2}$$

$$\sum_{(j,l)|(n,j,l,k)\in\Delta_{(n+1)k}^{-}} f_{njlk} = f_{(n+1)kt}, \quad \forall k \in N^{n+1}, \tag{4.3}$$

$$\sum_{k\in N^{n+1}} f_{(n+1)kt} = w, \tag{4.4}$$

$$x_{ij} = \sum_{(k,l)|<k,j,l>\in\delta_i} f_{ijkl}, \quad \forall i \in \{1, \ldots, n\}, j \in D_i, \tag{4.5}$$

$$f_{ijkl} \in \{0, 1\} \quad \forall i \in \{1, \ldots, n\}, < k, j, l > \in \delta_i, \tag{4.6}$$

$$f_{(n+1)kt} \in \{0, 1\} \quad \forall k \in N^{n+1}. \tag{4.7}$$

Constraints (4.5) link the *decision variables* x with the *flow variables*. Note that in the case where the `MIP regular` constraint is the only constraint in the model, the decision variables x and constraints (4.5) are not needed in the model. Without these, the resulting model is a pure network flow formulation that reduces to the determination of a path between s and t in an acyclic network, which can be solved very efficiently by a specialized algorithm (Ahuja *et al.* (1993)). In the case where the `MIP regular` constraint is part of a larger model, constraints (4.5) allow to formulate the rest of the model using the decision variables x.

The number of variables in the `MIP regular` model is $O(n|T|)$ where n is the sequence length and $|T|$ is the number of transitions in automaton A_1. The number of contraints is $O(n|Q|)$ where $|Q|$ is the number of states in automaton A_1.

Thus, introducing a `MIP regular` constraint to a MIP model induces the addition of a set of flow conservation linear constraints (4.1)-(4.4) and linking constraints (4.5) to the model. We use a procedure with the following signature:

$$\texttt{AddMIPRegular}(\Pi(Q,\Sigma,\delta,q_0,\, F),\, n,\, x,\, w,\, M),$$

to add the linear constraints associated with a `MIP regular` constraint to a model M, given a DFA Π, the decision variables x subject to the constraint, the length of the sequence n formed by these variables and the amount of flow w entering the graph.

The following lemma states that the set of solutions to constraints (4.1) to (4.4) corresponds to the set of words recognized by automaton A.

Lemma 1 *Let A be an automaton, as depicted in Figure 4.4, defined by a set of states and transitions. Let G be the graph associated with A, as depicted in Figure 4.5, defined by a set of nodes and arcs. A solution to the flow conservation equations (4.1) to (4.4) with $w = 1$ corresponds to a word recognized by automaton A.*

Proof: A solution f to the flow conservation equations (4.1) to (4.4) with $w = 1$ forms a path p_G in graph G. By construction, there is a corresponding path p_A in A starting from the initial state and ending at a final state. Following path p_A in automaton A is equivalent to process a word, letter by letter, by taking the transition labeled with the letter corresponding to the arc from path p_G at position $i = 1$, then $i = 2$, and so on until $i = n$. Since a final state is reached at the end of the process, this word is recognized by automaton A. Conversely, a word recognized by automaton A corresponds to a path in the automaton. By definition, for each transition in A, there is an arc in graph G associated with a 0-1 variable. We can build a solution f by setting to one the variables associated with the transitions along this path, as well as the variable linking the final state reached by the path to the sink node, and all the other variables to zero. This solution satisfies constraints (4.1) to (4.4) since it forms a path in graph G. ■

4.5 MIP Grammar

As we did with regular languages, we derive a MIP that accepts any sequence belonging to a context-free language. The model directly comes from the and/or graph presented in Section 4.3.2. Each node $N(A, i, j)$ in the and/or graph corresponds to a MIP 0-1 variable $X(A, i, j)$. A leaf node $N(t, i, 1)$ is associated to the MIP 0-1 decision variable x_{it}. The variable x_{it} is equal to one if and only if the node $N(t, i, 1)$ is *true*. A leaf node is considered as an or-node in the graph. In the following, the notations N_{or} and N_{and} refer to general or-node and and-node, while the notations X_{or} and X_{and} refer to their associated 0-1 variables. As we did for the `MIP regular` constraint, we introduce a 0-1 variable w that specifies if the constraint is active. When $w = 0$, every variable x_{it} must be assigned to zero.

The constraints on the variables of the MIP depend on the relationship between the corresponding nodes in the and/or graph. There is one significant difference between the MIP and the and/or graph. When there exist more than one parsing tree for a sequence, all nodes in the and/or graph that belong to at least one parsing tree are set to *true*, while for the MIP, one parsing tree is arbitrarily selected and all its variables are set to one. All other variables, including those that belong to other parsing trees, are set to zero. Choosing an arbitrary parsing tree simplifies the MIP without changing the solution space. Indeed, only one parsing tree is necessary to prove that a sequence belongs to a context-free language. We now present the constraints representing a MIP of a `grammar`.

Let N_{or} be an or-node other than a leaf node. Let $c(N_{or})$ be its children's label. The following constraint forces X_{or} to be equal to one if exactly one of the variables associated with the children of N_{or} is equal to one:

$$X_{or} = \sum_{n \in c(N_{or})} X_{and,n}. \tag{4.8}$$

A node belongs to a parsing tree only if exactly one of its parent belongs to the parsing tree. Let N_{or} be an or-node and $p(N_{or})$ be its parents label. We have the following equality:

$$X_{or} = \sum_{n \in p(N_{or})} X_{and,n}. \tag{4.9}$$

Note that these constraints imply those specifying that if an and-node N_{and} is true, then its children, $c(N_{and})$, are also true. Indeed, for each $m \in c(N_{and})$, we have $X_{or,m} = \sum_{n \in p(N_{or})} X_{and,n} \geq X_{and}$.

Finally, we force the root of the directed acyclic graph (DAG) to be assigned to one if and only if the constraint is active, i.e., when $w = 1$:

$$X(S, 1, n) \quad = \quad w. \tag{4.10}$$

The number of variables in the `MIP grammar` model is equal to the number of nodes in the graph, which is $O(n^3|G|)$ where n is the sequence length and $|G|$ is the number of productions in grammar G. The number of contraints is equal to twice the number of or-nodes which is $O(n^2|G|)$. The following lemma is a first step towards establishing the correspondance between a solution satisfying constraints (4.8)-(4.10) and a word recognized by the grammar.

Lemma 2 *Constraints (4.8), (4.9), and (4.10) are satisfied if and only if the and/or graph evaluates to* true*.*

Proof: Observe that all parents and children of an or-node are and-nodes and all parents and children of an and-node are or-nodes. And-nodes have a unique parent.

($\Longrightarrow$) Suppose there exists a valid assignment of every node in the and/or graph such that the root is assigned to *true*. Quimper and Walsh (2007) showed that there exists at least one parsing tree whose nodes in the graph are assigned to *true*. We arbitrarily select one such parsing tree and set to one every variable whose corresponding or-node belongs to the parsing-tree. Consider an or-node in the parsing tree. Among all its children assigned to *true*, we arbitrarily select one and-node and set its corresponding variable to one. All unassigned variables remaining in the MIP are set to zero. Constraint (4.8) is satisfied since an or-node is set to one if exactly one child is set to one. Constraint (4.9) is also satisfied since in the parsing tree, each node has only one parent (except for root). Finally, constraint (4.10) is satisfied since the root node is set to one.

($\Longleftarrow$) Suppose the MIP is feasible. For every variable assigned to one, we assign the corresponding node to *true*. By constraint (4.8), every or-node has one child set to *true*. By constraint (4.9), every or-node has one parent set to *true*. This parent is an and-node that has either one or two children. In either case, constraint (4.9)

ensures that a node that has a parent set to *true* is also set to *true*. Therefore, the children of an and-node set to *true* are also set to *true*. Finally, by constraint (4.10), the root node of the tree is set to *true*. We proved that every variable set to one in the MIP have its corresponding node set to *true* in the graph. We now prove that values set to zero in the MIP can be assigned to Boolean values in the graph. We set to *false* every leaf node whose corresponding MIP variable is set to zero. Every unassigned node is evaluated using a bottom-up approach, i.e., and-nodes are assigned to *true* if both children are *true* and or-nodes are assigned to *true* if at least one child is *true*. Therefore, we obtain a valid assignment of Boolean values in the and/or graph. ■

Since Quimper and Walsh (2007) proved that the grammar recognizes a sequence if and only if the and/or graph evaluates to *true*, we conclude that the grammar accepts only the sequences satisfying the MIP.

Theorem 1 *Constraints (4.8), (4.9), and (4.10) are satisfied if and only if the grammar recognizes the sequence s for which $X(s[i], i, 1) = 1$.*

4.6 Comparison Between MIP Regular and MIP Grammar

It is interesting to compare `MIP regular` obtained from an automaton Π with the `MIP grammar` obtained from a context-free grammar G that encodes the same language recognized by Π. We recall how to automatically generate the grammar G from the transitions of the automaton Π. The states of the automaton form the set of non-terminals of the grammar and the alphabet symbols form the set of terminals. The starting non-terminal of the grammar is the initial state of the automaton. Each transition in the automaton is associated with a production in the grammar as follows. If S_1 is the state at the beginning of a transition, S_2 is the state at the end of this transition, and α is the associated alphabet symbol, then the production $S_1 \rightarrow \alpha S_2$ is added to the grammar. If S_2 is a final state then the production $S_1 \rightarrow \alpha$ is also added to the grammar.

The and/or graph produced with such a grammar has the following properties. The only productions having two literals on their right-hand side have the form $S_1 \rightarrow \alpha S_2$, i.e., a non-terminal produces a terminal followed by a non-terminal. The parsing trees produced by this grammar are therefore unbalanced trees where the left child

of a node is necessarily a leaf labeled with a terminal symbol. For instance, the production $S_1 \rightarrow \alpha S_2$ creates in a parsing tree a node S_1 with a left-child α and a right-child S_2. The or-nodes in the graph created from a sequence of n characters are either of the form $N(P, t, n-t+1)$ for the inner-nodes or the form $N(\alpha, t, 1)$ for the leaf nodes.

The and/or graph associated to a regular grammar has a similar structure to the layered graph used to model the `MIP regular`. The or-node $N(P, t, n-t+1)$ in the and/or graph corresponds to the node $P \in N^t$ in the layered graph of the `MIP regular`. An and-node with left child $N(\alpha, t, 1)$, right child $N(Q, t+1, n-t)$ and parent $N(P, t, n-t+1)$ corresponds to the arc (t, α, P, Q) in the layered graph of the `MIP regular`. Figure 4.6 shows the relation between the arc $(4, b, 2, 3)$ from the layered graph of `MIP regular` depicted in Figure 4.5 and the corresponding nodes in the and/or graph.

The similarities between both graphs lead to similarities in the corresponding MIPs. Let $O_{t,A}$ be the 0-1 variable associated to the inner node $N(P, t, n-t+1)$ and $x_{t,\alpha}$ be the 0-1 variable associated to the leaf node $N(\alpha, t, 1)$. Let $A_{t,P,Q,\alpha}$ be the 0-1 variable associated to an and-node whose parent is $N(P, t, n-t+1)$, whose left child is the leaf node $N(\alpha, t, 1)$, and whose right child is the node $N(Q, t+1, n-t)$. Figure 4.6 shows some nodes in an and/or graph and their corresponding variables.

Equations (4.8) and (4.9), when applied on the inner node $N(P, t, n-t+1)$, lead to the two following equations:

$$O_{t,P} = \sum_{Q,\alpha} A_{t,P,Q,\alpha}, \tag{4.11}$$

$$O_{t,P} = \sum_{Q,\alpha} A_{t+1,Q,P,\alpha}. \tag{4.12}$$

These two equations lead to the following one, which is strictly equivalent to the flow conservation constraint (4.2) in the `MIP regular`:

$$\sum_{Q,\alpha} A_{t,P,Q,\alpha} = \sum_{Q,\alpha} A_{t+1,Q,P,\alpha}. \tag{4.13}$$

Equation (4.9), when applied on the leaf node $N(P, t, 1)$, gives the following equa-

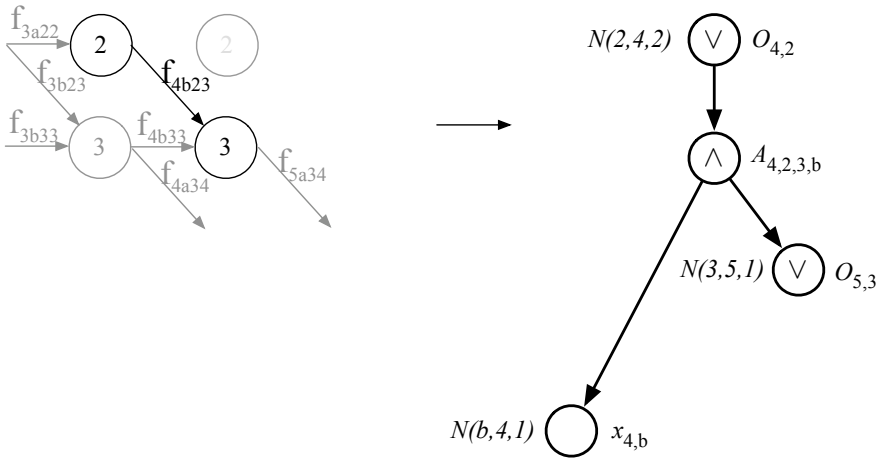

FIGURE 4.6 The relation between the arc $(4, b, 2, 3)$ from the layered graph of `MIP regular` depicted in Figure 4.5 and the corresponding nodes in the and/or graph. The label of each node is written on the left and the associated 0-1 variable on the right.

tion, which is equivalent to constraint (4.5) in the `MIP regular`:

$$x_{t,\alpha} \;=\; \sum_{P,Q} A_{t+1,P,Q,\alpha}. \tag{4.14}$$

Constraints (4.8) and (4.10) on the root node $N(S, 1, n)$ lead to the following equations, which are equivalent to constraint (4.1):

$$w = \;\; O_{1,S} \;= \sum_{Q,\alpha} A_{1,S,Q,\alpha}. \tag{4.15}$$

Therefore, we have shown the following result:

Theorem 2 *The `MIP grammar` model resulting from a grammar encoding a regular language is equivalent to the `MIP regular` model encoding the same language. Moreover, the LP relaxations of the two models are also equivalent.*

The equivalence of the LP relaxations follows from the fact that the equivalence between the two MIP models is derived from a series of linear equations which are true whether the variables are binary or continuous. Although both the MIP models and their LP relaxations provide the same optimal values, the two models show some differences. For instance, notice that we have a 0-1 variable for each or-node in the

`MIP grammar`, while there is no such variable associated to the states of the layered graph in the `MIP regular`.

4.7 Case Study

To evaluate the quality of our modeling approaches for constrained sequences of decision variables, we present computational results on complete shift scheduling problems described in Demassey *et al.* (2005). We compare our models to a compact assignment model. Since implicit formulations are limited to modeling single work activity problems, we did not experiment with such models. We did not implement explicit formulations neither, as the number of variables would be excessive (for instance, there are 67,752,783 variables for two work activities in our case study).

4.7.1 Problem Definition

The benchmarks are randomly generated, but are based on rules from a real-world shift scheduling problem. The demand curves come from a retail store. The objective is to create an optimal employee schedule for one day that satisfies the work regulation rules and the demands for each work activity.

The one day planning horizon is decomposed into 96 periods of 15 minutes each. We introduce the following notations before we define the problem:

- E : set of available employees;
- W : set of work activities;
- J : set of all activities ($J = W \cup \{l, p, o\}$)
  where l = lunch, p = break, o = rest;
- $I = \{1, 2, \ldots, n\}$: set of periods. $I' = I \setminus \{1\}$;
- $F_i \subseteq J$: set of activities that are not allowed to be performed at period $i \in I$;
- c_{ij}: cost for an employee to cover an activity $j \in W \setminus F_i$ at period $i \in I$.

Work Regulation Rules

1. Activities $j \in F_i$ are not allowed to be performed at period $i \in I$.
2. If an employee is working, he must cover between 3 hours and 8 hours of work activities.

3. If a working employee covers at least 6 hours of work activities, he must have two 15 minute breaks and a lunch break of 1 hour.
4. If a working employee covers less than 6 hours of work activities, he must have a 15 minute break, but no lunch.
5. If performed, the duration of any activity $j \in W$ is at least 1 hour (4 consecutive periods).
6. A break (or lunch) is necessary between two different work activities.
7. Work activities must be inserted between breaks, lunch and rest stretches.
8. Rest activities have to be assigned at the beginning and at the end of the day.

Demand Covering

1. The required number of employees for activity $j \in W \setminus F_i$ at period $i \in I$ is d_{ij}. Undercovering and overcovering are allowed. The cost of undercovering activity $j \in W \setminus F_i$ at period $i \in I$ is c_{ij}^- by unit of undercovering and the cost of overcovering activity $j \in W \setminus F_i$ at period $i \in I$ is c_{ij}^+ by unit of overcovering.

The following sections present four ways of modeling this problem. The first model is a compact assignment MIP formulation that does not exploit the `MIP regular` constraint nor the `MIP grammar` constraint. The second model uses the `MIP regular` constraint and the third and fourth, the `MIP grammar` constraint.

4.7.2 A Compact Assignment MIP Model

Decision Variables

$$x_{eij} = \begin{cases} 1, & \text{if employee } e \in E \text{ covers activity } j \in J \text{ at period } i \in I, \\ 0, & \text{otherwise.} \end{cases}$$

Work Regulation Rules

Rule 1:

$$x_{eij} = 0, \qquad e \in E, i \in I, j \in F_i. \tag{4.16}$$

Rule 2:

$$w_e = \begin{cases} 1, & \text{if employee } e \in E \text{ is working,} \\ 0, & \text{otherwise.} \end{cases}$$

$$\sum_{j \in J} x_{eij} = w_e, \qquad e \in E, i \in I, \tag{4.17}$$

$$12w_e \leq \sum_{i \in I} \sum_{j \in W \setminus F_i} x_{eij} \leq 32w_e, \qquad e \in E. \tag{4.18}$$

Rules 3 and 4:

$$u_e = \begin{cases} 1, & \text{if employee } e \text{ covers at least 6 hours of work activities,} \\ 0, & \text{otherwise.} \end{cases}$$

$$\sum_{i \in I} \sum_{j \in W \setminus F_i} x_{eij} - 8u_e \leq 24, \qquad e \in E, \tag{4.19}$$

$$\sum_{i \in I} \sum_{j \in W \setminus F_i} x_{eij} \geq 23u_e, \qquad e \in E, \tag{4.20}$$

$$\sum_{i \in I} x_{eip} = u_e + w_e, \qquad e \in E, \tag{4.21}$$

$$\sum_{i \in I} x_{eil} = 4u_e, \qquad e \in E. \tag{4.22}$$

Rule 5:

$$v_{eij} = \begin{cases} 1, & \text{if employee } e \in E \text{ starts activity } j \in W \text{ at period } i \in I, \\ 0, & \text{otherwise.} \end{cases}$$

$$v_{eij} \geq x_{eij} - x_{e(i-1)j}, \qquad e \in E, i \in I, j \in W \setminus F_i \cup \{o\}, \tag{4.23}$$

$$v_{eij} \leq x_{eij}, \qquad e \in E, i \in I, j \in W \setminus F_i \cup \{o\}, \tag{4.24}$$

$$v_{eij} \leq 1 - x_{e(i-1)j}, \qquad e \in E, i \in I, j \in W \setminus F_i \cup \{o\}, \tag{4.25}$$

$$x_{ei'l} \geq v_{eil}, \qquad e \in E, i \in I, i' = i, i+1, i+2, i+3, \tag{4.26}$$

$$x_{ei'j} \geq v_{eij}, \qquad e \in E, i \in I, i' = i, i+1, i+2, i+3, j \in W \setminus F_i. \tag{4.27}$$

Rule 6:

$$v_{eij} \leq 1 - \sum_{j' \in W \setminus F_{i-1}} x_{e(i-1)j'}, \qquad e \in E, i \in I', j \in W \setminus F_i. \tag{4.28}$$

Rule 7:

$$x_{eip} \leq 1 - x_{e(i-1)p}, \qquad e \in E, i \in I', \tag{4.29}$$

$$x_{eip} \leq \sum_{j \in W \setminus F_{i-1}} x_{e(i-1)j}, \qquad e \in E, i \in I', \tag{4.30}$$

$$x_{eip} \leq \sum_{j \in W \setminus F_{i+1}} x_{e(i+1)j}, \qquad e \in E, i \in I', \tag{4.31}$$

$$v_{eil} \leq 1 - x_{e(i-1)p}, \qquad e \in E, i \in I', \tag{4.32}$$

$$v_{eil} \leq \sum_{j \in W \setminus F_{i-1}} x_{e(i-1)j}, \qquad e \in E, i \in I', \tag{4.33}$$

$$v_{eil} \leq \sum_{j \in W \setminus F_{i+1}} x_{e(i+3)j}, \qquad e \in E, i \in I'. \tag{4.34}$$

Rule 8:

$$v_{ei}^{-} = \begin{cases} 1, & \text{if employee } e \in E \text{ covers at least one working activity} \\ & \text{beginning before period } i \in I; \\ 0, & \text{otherwise.} \end{cases}$$

$$v_{ei}^{+} = \begin{cases} 1, & \text{if employee } e \in E \text{ covers at least one working activity} \\ & \text{beginning after period } i \in I; \\ 0, & \text{otherwise.} \end{cases}$$

$$v_{ei}^{-} \leq \sum_{i^{-}<i} \sum_{j \in W \setminus F_{i^{-}}} v_{ei^{-}j}, \qquad e \in E, i \in I', \tag{4.35}$$

$$v_{ei}^{-} \geq \sum_{j \in W \setminus F_{i^{-}}} v_{ei^{-}j}, \qquad e \in E, i \in I, i^{-} < i, \tag{4.36}$$

$$v_{ei}^{+} \leq \sum_{i^{+}>t} \sum_{j \in W \setminus F_{i^{+}}} v_{ei^{+}j}, \qquad e \in E, i \in I', \tag{4.37}$$

$$v_{ei}^{+} \geq \sum_{j \in W \setminus F_{i^{+}}} v_{ei^{+}j}, \qquad e \in E, i \in I, i^{+} > i, \tag{4.38}$$

$$x_{eio} \leq (1 - v_{ei}^{-}) + (1 - v_{ei}^{+}), \qquad e \in E, i \in I. \tag{4.39}$$

Demand Covering

$$\sum_{e \in E} x_{eij} - s_{ij}^{+} + s_{ij}^{-} = d_{ij}, \qquad i \in I, j \in W \setminus F_t. \tag{4.40}$$

Objective Function

$$\min \qquad \sum_{i \in I} \sum_{j \in W \setminus F_i} (\sum_{e \in E} c_{ij} x_{eij} + c_{ij}^{+} s_{ij}^{+} + c_{ij}^{-} s_{ij}^{-}). \tag{4.41}$$

4.7.3 A MIP Regular Model

To observe the impact of modeling with the `MIP regular` constraint, we include several rules of the problem in a DFA and we formulate the other rules and the objective function as stated in Section 4.7.2. Work regulation rules 1 to 4 and demand covering constraints are formulated as in the compact assignment model, and work regulation rules 5 to 8 are included in the DFA. We use the DFA suggested by Demassey *et al.* (2006) for the same problem. The DFA presented in Figure 4.7 is for the problem with two work activities (a and b on the figure). By following the arcs from the initial state on the left side to the final states, one obtains a shift that alternates between rest periods, sequences of activities of at least 4 periods, breaks and lunch periods. It is easily generalized for any number of work activities. Let us denote Π_n the DFA for the problem with n work activities.

We insert a `MIP regular` constraint for each employee $e \in E$ to the model. This constraint ensures that the covering of the activities $j \in W$ for each $i \in I$ for any

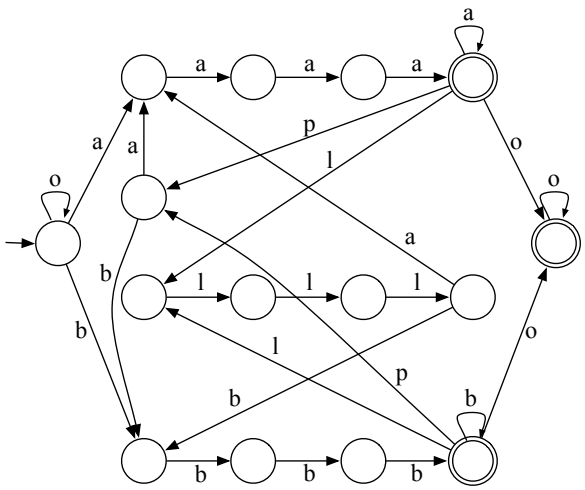

FIGURE 4.7 DFA Π_2 for two activities

employee $e \in E$ is a word recognized by the DFA $\Pi_{|W|}$. To add this constraint, we use the procedure presented in Section 4.4:

$$\textbf{AddMIPRegular}(\Pi_{|W|}, |I|, x_e, w_e, M), \qquad \forall e \in E, \tag{4.42}$$

where M is the model presented in the previous section without work regulation constraints 5 to 8, and the variables x_e and w_e have the same interpretation as in this model.

4.7.4 MIP Grammar Models

To test the `MIP Grammar` constraint on this problem, we tried two different grammars. First, we used a grammar encoding the DFA presented in Section 4.7.3 with the same linear constraints as in the two previous models for work regulation rules 1 to 4 and the demand constraints. We call this model the *partial* `MIP grammar` *model*. This grammar is obtained from the automaton Π as described in Section 4.6.

Then, we used a context-free grammar presented in Quimper and Walsh (2007) that encodes all work regulation rules. It only uses the demand constraints as side constraints. This second grammar, leading to the *complete* `MIP grammar` *model*, can be expressed as follows (for the sake of clarity, the grammar presented here is not in Chomsky normal form, but note that any context-free grammar can be converted to

Chomsky normal form (Hopcroft *et al.* (2001))):

$$R \rightarrow oR \mid o \qquad L \rightarrow lL \mid l$$
$$A \rightarrow aA \mid a \qquad G \rightarrow A$$
$$P \rightarrow GpG \qquad Q \rightarrow PpG$$
$$F \rightarrow PLP \mid QLG \mid GLQ \qquad S \rightarrow RPR \mid RFR$$

where R, A, P, F, L, G, Q are non-terminals, S is the starting non-terminal. Terminals o, p, l represent rest, break and lunch periods respectively. Terminal a represents a work activity.

Several rules of the model are handled with a set of restrictions on the productions of the grammar. These rules are dealt within the construction of the and/or graph. As for the `MIP regular` model, a `MIP grammar` constraint is posted for each available employee on a sequence length of $|I|$ in both models.

4.7.5 Computational Results

Experiments were run on a 2.4 GHz Dual AMD Opteron Processor 250 (where only one processor was used) with 3 GB of RAM, using the MIP solver CPLEX 10.0. Tables 4.1 to 4.4 present the results for the four formulations presented in the last sections for the problem with one work activity and 12 available employees (10 instances). In the tables, the $Time$ and $Value$ on the LP refer to the results for the LP relaxation of the models. These results were obtained without CPLEX presolve to allow comparing the LP relaxation lower bounds of the different models. The symbol "$>$" means that CPLEX could not find the solution of the LP relaxation within a time limit of 3600 seconds. The results on the MIP were obtained with CPLEX default parameters on the given models. In particular, the branch-and-bound algorithm is stopped when the objective value is within 1% of optimality, which explains the differences in the MIP objective values of two models for which CPLEX stops within the 3600 seconds elapsed time limit. The symbol "$>$" means that no integer solution was found within this time limit. $|V|$ and $|C|$ are, respectively, the number of variables and the number of constraints in the MIP model after CPLEX presolve.

Table 4.5 shows the results for the two work activity instances for the three formal language based models, since the compact assignment MIP formulation did not

succeed in finding any integer solution on these instances. In Table 4.5, $|C|$ and $|V|$ are, respectively, the number of variables and the number of constraints in the MIP model after CPLEX presolve. *Time* is the time needed by CPLEX to find the optimal solution for the models. The symbol ">" means that CPLEX could not find the optimal integer solution within the time limit of 3600 seconds. *Gap* is the gap between the initial LP lower bound, Z_{LP}, and the value of the best integer solution found Z_{MIP} at the end of the solving process:

$$Gap = \left(\frac{Z_{MIP} - Z_{LP}}{Z_{LP}}\right) \times 100.$$

The symbol ">" in the *Gap* column means that no integer solution was found within the time limit.

Note that the three models using formal languages lead to a faster computation than the compact assignment MIP model on all instances in our benchmarks. In particular, the multi-activity case cannot be handled by the compact assignment model.

The LP relaxation bounds for the `MIP regular` and the partial `MIP grammar` models are the same, as explained in Section 4.6. However, the computation times between the two models are often quite different. Globally, it seems that the `MIP regular` model leads to faster computations than the partial `MIP grammar` model. Observe that the LP relaxation bounds of these two formulations are never worse than the LP bounds of the compact assignment model, and are stronger on 4 out of the 10 instances.

The `MIP regular` and the partial `MIP grammar` models both lead to faster computation time than the complete `MIP grammar` model. However, the idea of having all the constraints of a problem in a single structure such as the and/or graph can lead to specific solution methods like a local search framework Quimper and Rousseau (2009).

Some constraint programming based approaches addressed the problem studied here. The results on the `MIP regular` models are competitive with those of the branch-and-price approach described in Demassey *et al.* (2006). Indeed, as reported in Demassey *et al.* (2006), for the one work-activity problems, 8 instances are solved to optimality by the branch-and-price with an average of 144 seconds of computation time, while for the two work-activity problems, 8 instances are solved to optimality

TABLE 4.1 Results for the compact assignment MIP model

	LP				MIP	
Id	*LP Value*	*LP Time*	$\|C\|$	$\|V\|$	*MIP Value*	*MIP Time*
1	138,8952	36,53	29871	4040	172,6670	2142,17
2	162,9396	104,73	56743	5104	>	>
3	168,8223	89,06	56743	5104	>	>
4	131,6560	32,95	45983	4704	152,2240	3610,00
5	143,7443	63,61	40447	4360	171,9930	3610,01
6	129,0947	36,65	40447	4360	137,5180	3616,57
7	148,4681	38,26	45887	4608	>	>
8	147,2002	90,49	56743	5104	>	>
9	142,4836	27,15	36175	4156	182,5370	3607,76
10	145,9563	36,89	45983	4704	149,1810	3611,72

TABLE 4.2 Results for the `MIP regular` model

	LP				MIP	
Id	*LP Value*	*LP Time*	$\|C\|$	$\|V\|$	*MIP Value*	*MIP Time*
1	138,8952	15,95	1491	1856	172,6670	1,03
2	162,9396	13,32	2719	3976	164,1370	40,09
3	168,8223	15,70	2719	3976	169,0120	64,64
4	131,6560	19,15	2183	3144	133,3830	46,39
5	144,4182	30,81	1915	2728	145,4640	14,03
6	133,0766	12,34	1915	2728	135,2180	3,28
7	149,2739	26,55	2183	3144	150,6810	5,99
8	147,2002	12,81	2719	3976	148,0470	131,77
9	142,4836	11,84	1759	2416	182,5370	16,14
10	146,2410	23,66	2183	3144	147,5030	20,22

TABLE 4.3 Results for the partial `MIP grammar` model

Id	LP		MIP			
	LP Value	*LP Time*	$\|C\|$	$\|V\|$	*MIP Value*	*MIP Time*
1	138,8952	34,63	1683	2060	172,6665	1,16
2	162,9396	31,67	3040	4285	163,9210	66,26
3	168,8223	41,62	3040	4285	170,5663	46,74
4	131,6560	42,89	2459	3408	133,1414	98,00
5	144,4182	52,52	2143	2956	145,2850	21,83
6	133,0766	57,56	2143	2956	135,1286	1,56
7	149,2739	44,03	2456	3405	150,7675	53,82
8	147,2002	47,63	3040	4285	148,0467	263,43
9	142,4836	27,46	1957	2614	182,4833	12,91
10	146,2410	37,19	2458	3408	147,6853	19,28

TABLE 4.4 Results for the complete `MIP grammar` model

Id	LP		MIP			
	LP Value	*LP Time*	$\|C\|$	$\|V\|$	*MIP Value*	*MIP Time*
1	>	3612,35	2727	6356	172,6665	7,42
2	>	3610,68	35419	190480	>	>
3	>	3615,64	35419	190480	>	>
4	>	3614,92	19163	84672	133,0630	1850,38
5	>	3609,11	10387	41464	145,8830	322,57
6	>	3618,18	10387	41464	134,8178	130,21
7	>	3618,83	19163	84672	151,2082	1662,75
8	>	3611,39	35419	190480	>	>
9	>	3614,71	5503	21580	182,5370	1015,10
10	>	3610,10	19163	84672	147,0870	1313,28

TABLE 4.5 Results for the two work activity instances

	MIP regular				Partial MIP grammar				Complete MIP grammar			
Id	$\|C\|$	$\|V\|$	*Time*	*Gap*	$\|C\|$	$\|V\|$	*Time*	*Gap*	$\|C\|$	$\|V\|$	*Time*	*Gap*
1	3111	5084	228,07	0,00	2727	4436	>	>	8570	25363	2826,40	0,00
2	3779	6192	2870,20	0,99	3299	5472	277,48	1,00	18634	87215	1952,58	0,00
3	4475	6412	1541,15	0,74	3643	5656	464,27	0,74	28938	185427	>	>
4	3571	7512	169,96	0,80	3863	6552	>	3,05	21298	92195	>	>
5	3879	5668	>	11,52	3223	4804	>	99,99	24174	144327	>	>
6	3199	6116	1288,56	0,43	3483	5408	82,72	0,08	22646	118099	>	>
7	5799	4972	29,94	0,41	2839	4336	>	53,85	8874	33195	>	>
8	4595	9740	>	4,96	5019	8456	2731,12	0,00	39782	237691	325,08	0,70
9	4595	7680	>	3,64	3971	6624	>	6,11	28042	146291	>	>
10	4667	7584	1108,23	0,87	4127	6756	58,12	0,89	34846	229475	>	100,00

by the same approach with an average execution time of 394 seconds. Menana and Demassey (2009) succeed in finding the lowest cost schedule for one employee for up to 50 work activities. Kadioglu and Sellmann (2008) tested an incremental arc-consistency algorithm for context-free grammars on a simplified version of the problem presented here, with the objective of minimizing the number of employees, without taking into account overcosts and undercosts. They found the optimal solution for the 10 one-activity instances with an average of 9 seconds of computation time and for 7 out of 10 two-activity instances with an average of 9 seconds of computation time as well.

4.8 Conclusion

We presented two new MIP modeling approaches to express constraints on sequences of decision variables. These approaches are inspired by two CP constraints using formal languages: the `regular` constraint and the `grammar` constraint. The MIP version of the `regular` constraint uses an automaton to model the rules on the sequence of decision variables and transform it into a set of linear constraints representing a network flow problem in the graph. The MIP version of the `grammar` constraint uses an and/or decomposition of the parsing tree of all the sequences accepted by a context-free grammar and translates the logical clauses associated with

the graph into linear constraints on 0-1 variables.

From a modeling point of view, both approaches allow the design of complex rules on sequences of variables to be handled with formal languages tools (an automaton or a context-free grammar) instead of directly into linear constraints. This process generates automatically a set of linear constraints that can be managed by any MIP solver. With this approach, the modeling of many complex rules is simplified and experimental results show that the resulting formulations can be strong.

Despite its interesting structure, the complete `MIP grammar` models do not lead to competitive computation times. We will explore two research avenues to address this issue. First, in a context where all employees are identical, we will study an implicit grammar based model using a single and/or graph to represent all employees, which reduces significantly the model size and allows us to tackle multi-activity problems. Also, we are looking into the use of complete `MIP grammar` models as subproblems of a column generation framework to handle large-scale problems, including those where employees must be distinguished.

Acknowledgments

We thank two anonymous referees for their constructive comments which have helped us to improve the paper significantly. Funding for this project was provided by the Natural Sciences and Engineering Council of Canada (NSERC) and by the Fonds québécois de recherche sur la nature et les technologies (FQRNT Québec). This support is gratefully acknowledged.

Chapitre 5

GRAMMAR-BASED INTEGER PROGRAMMING MODELS FOR MULTI-ACTIVITY SHIFT SCHEDULING

Dans la littérature, un grand nombre de travaux aborde les problèmes de planification de quarts en supposant que les employés disponibles sont tous identiques. Cette supposition a permis le développement de modèles implicites pour pallier aux modèles de type recouvrement qui font intervenir rapidement un trop grand nombre de variables pour être résolus directement. Par contre, les modèles implicites existant dans la littérature ne font intervenir que quelques formes de flexibilité, notamment dans le positionnement des pauses, mais, à notre connaissance, aucun d'entre eux ne peut se généraliser directement à des cas où plusieurs activités de travail sont prises en compte.

L'utilisation de grammaires hors-contextes pour la modélisation de règles régissant la composition de quarts de travail amène une façon expressive d'aborder les problèmes de planification de quarts. Dans le présent article, nous proposons un modèle implicite utilisant les grammaires de manière à pouvoir traiter les problèmes de planification de quart de personnel anonymes aux contraintes complexes, dont les problèmes multi-activités.

Le modèle implicite est une extension du modèle en nombres entiers 0-1 basé sur les grammaires introduit au chapitre précédent. À partir d'une grammaire, le graphe orienté acyclique associé est créé et une série de contraintes linéaires est dérivée de ce graphe. Plutôt que d'utiliser des variables binaires, les variables du modèle implicite sont des variables entières positives et leurs valeurs dans une solution représentent le nombre d'arbres d'analyse auxquels ils font partie. Chaque arbre d'analyse étant

associée à un quart, une solution au modèle implicite est un ensemble de quarts satisfaisant la demande qui peut être identifié au moyen d'une procédure simple.

Nous démontrons que le modèle implicite basé sur les grammaires a la même relaxation linéaire que le modèle de recouvrement de Dantzig et que les principaux modèles implicites présents dans la littérature.

Finalement, nous comparons expérimentalement le modèle implicite sur deux classes de problèmes de planification de quarts anonymes : des instances mono-activités et des instances multi-activités. La première classe d'instances montre que notre modèle permet de modéliser une variété de problèmes mono-activité et se comparer expérimentalement à des modèles de la littérature. Les comparaisons sur des problèmes multi-activités démontrent plus clairement l'intérêt de notre approche de modélisation. Le modèle implicite basé sur les grammaires est le seul à pouvoir modéliser les problèmes comportant jusqu'à 10 activités de travail et être résolu efficacement par un logiciel commercial.

L'article suivant a été soumis à la revue *Management Science* en décembre 2009, puis une version révisée a été soumise en juin 2010.

Grammar-Based Integer Programming Models for Multi-Activity Shift Scheduling

Marie-Claude Côté

Département de mathématiques et génie industriel, École Polytechnique de Montréal, PO Box 6079, succ. Centre-Ville, Montréal, H3C 3A7, Canada,
CIRRELT-Interuniversity Research Center on Enterprise Networks, Logistics and Transportation, Montréal, Canada, marie-claude.cote@polymtl.ca

Bernard Gendron

Département d'informatique et de recherche opérationnelle, Université de Montréal,Pavillon André-Aisenstadt, PO Box 6128, succ. Centre-Ville, Montréal, H3C 3J7, Canada,
CIRRELT-Interuniversity Research Center on Enterprise Networks, Logistics and Transportation, Montréal, Canada Bernard.Gendron@cirrelt.ca

Louis-Martin Rousseau

Département de mathématiques et génie industriel, École Polytechnique de Montréal, PO Box 6079, succ. Centre-Ville, Montréal, H3C 3A7, Canada,
CIRRELT-Interuniversity Research Center on Enterprise Networks, Logistics and Transportation, Montréal, Canada louis-martin.rousseau@polymtl.ca

Abstract

This paper presents a new implicit formulation for shift scheduling problems, using context-free grammars to model regulation in the composition of shifts. From the grammar, we generate an integer programming (IP) model having a linear programming (LP) relaxation equivalent to that of Dantzig's set covering model. When solved by a state-of-the-art IP solver on problem instances with a small number of shifts, our model, the set covering formulation and a typical implicit model from the literature yield comparable solution times. On instances with a large number of shifts, our formulation shows superior performance and can model a wider variety of constraints. In particular, multi-activity cases, which cannot be modeled by existing implicit formulations, can easily be handled with grammars. We present comparative experimental results on a large set of instances involving one work-activity, as well as on problems dealing with up to ten work-activities.

5.1 Introduction

In this paper, we consider *shift scheduling problems* defined over a planning horizon of one day, divided into multiple periods. In this context, a shift is defined by its starting and ending times and by the activities or breaks to be performed at each period. The assignment of activities to a shift is constrained by different rules mainly arising from work regulation agreements and ergonomic considerations. In a *single-activity* shift scheduling problem, one only specifies, at each period, if an employee is working or not. In a *multi-activity* shift scheduling problem, there are several work-activities and whenever an employee is working at a given period, it is further necessary to specify which work-activity is assigned to that employee. In this paper, we deal with multi-activity shift scheduling problems in which all employees are identical.

The problem we consider is defined as follows. Given a planning horizon I divided into periods of equal length, a set of work-activities J, the set of all feasible shifts Ω, and the number of employees b_{ij} required at each period $i \in I$ for each work-activity $j \in J$, one must select from Ω a subset and multiplicities for each shift in this subset that covers the required number of employees at minimum cost. Each feasible shift $s \in \Omega$ has an associated cost $c_s \geq 0$, which we assume to be decomposable by period and by work-activity as follows: $c_s = \sum_{i \in I} \sum_{j \in J} \delta_{ijs} c_{ij}$, where $c_{ij} \geq 0$, for each $i \in I$ and $j \in J$, and $\delta_{ijs} = 1$ if work-activity $j \in J$ is assigned to period $i \in I$ in shift $s \in \Omega$.

We will consider two types of integer programming (IP) models for this problem, explicit and implicit, a terminology that is also used to characterize formulations for single-activity shift scheduling problems. In an *explicit* model, one obtains the schedule for each employee simply by scanning the optimal solution (i.e., in a time linear to the model size), while in an *implicit* model, a post-processing algorithm must be called upon in order to derive the schedule for each employee. This algorithm is typically efficient, especially when compared to solving the model, but its running time is generally not linear in the model size.

The following IP model, denoted D, extends in a straightforward manner the original set covering formulation proposed in Dantzig (1954) for the shift scheduling problem first described in Edie (1954). This model uses a variable x_s for each shift

$s \in \Omega$ that gives the number of employees assigned to shift s:

$$f(D) = \min \sum_{s \in \Omega} c_s x_s$$

$$\sum_{s \in \Omega} \delta_{ijs} x_s \geq b_{ij}, \qquad \forall i \in I, j \in J, \tag{5.1}$$

$$x_s \geq 0 \text{ and integer}, \qquad \forall s \in \Omega. \tag{5.2}$$

Model D explicitly enumerates all feasible shifts; therefore, we will call it the *shift-based explicit model*. Note that model D allows the formulation of problems with any cost structures that decompose by shift, not only by period and by work-activity, as we assume in our problem definition (see the Conclusion for a discussion on more general cost structures).

In practice, such explicit models can only be solved when Ω is relatively small, or else, by using column generation approaches as in Demassey *et al.* (2006); Mehrotra *et al.* (2000). In this paper, we present a new implicit formulation based on assignment variables y_{ij} indicating the number of employees assigned to activity $j \in J$ at period $i \in I$. These variables are related to the variables in the shift-based explicit model by the simple equations $y_{ij} = \sum_{s \in \Omega} \delta_{ijs} x_s$. More precisely, the nonnegative integer variables x_s defined over feasible shifts $s \in \Omega$ in model D are represented equivalently by using an additional set of integer variables $v \geq 0$ such that $(y, v) \in H$, where H is a bounded polyhedron. The implicit model that we propose, denoted Q, has therefore the following form:

$$f(Q) = \min \sum_{i \in I} \sum_{j \in J} c_{ij} y_{ij}$$

$$y_{ij} \geq b_{ij} \text{ and integer}, \qquad \forall i \in I, j \in J, \tag{5.3}$$

$$(y, v) \in H, \tag{5.4}$$

$$v \geq 0 \text{ and integer}. \tag{5.5}$$

Côté *et al.* (2007) and Côté *et al.* (2009) exploit automata and context-free grammars to formulate similar IP models that represent all feasible shifts for any single employee. Since they explicitly represent the assignment of work-activities to each employee, these formulations belong to the class of explicit models, but unlike model D, they do not use the set of shifts; hence, we will call them *employee-based explicit*

models. When the number of employees or activities increase, these employee-based explicit models do not scale well as performance degrades rapidly, mainly due to symmetry issues (see Section 5.5.2 for experimental results). In this paper, we show how to derive grammar-based models with tractable size, allowing us to handle large-scale problems with multiple activities. Assuming that any feasible shift can be represented by a word in a context-free language, we will show how to derive polyhedron H from the context-free grammar G defining the language. Moreover, we will show that polyhedron H is integral, and therefore that Q and D have equivalent linear programming (LP) relaxations. To the best of our knowledge, our approach is the first implicit modeling technique that is able to accurately formulate and efficiently solve *multi-activity* shift scheduling problems.

The remainder of the paper is organized as follows. In the next section, we present a literature review on shift scheduling problems and we introduce formal languages and grammar theory. In Section 5.3, we describe our modeling methodology using grammars to formulate shift scheduling problems. In Section 5.4, we present theoretical results relevant to our model; in particular, we demonstrate that our model has the same LP relaxation as Dantzig's set covering model. In Section 5.5, we present comparative computational results on classical shift scheduling problems from Mehrotra *et al.* (2000) and on a set of large-scale problem instances with multiple activities.

5.2 Background Material

This section reviews the literature on shift scheduling problems and presents basic notions of context-free grammars which are relevant to our study.

5.2.1 Shift Scheduling

For many organizations, finding the best schedule satisfying all their requirements and constraints is an important, but difficult task. Consequently, several studies were dedicated to this problem. Ernst *et al.* (2004a,b) present an exhaustive overview of models and methods for problems related to staff scheduling and rostering.

Implicit formulations provide an interesting alternative to the explicit model D. These models do not assign breaks to shifts a priori. Rather, they introduce the notion of *shift types*, which are characterized only by starting and ending times, giving no

details about how breaks are assigned within the shifts. Typically, these models capture the number of employees assigned to each shift type and to each break with different sets of variables. From an optimal solution to such an implicit model, one can retrieve the number of employees assigned to each shift type and each break, and construct an optimal set of shifts through a polynomial-time procedure.

Rekik *et al.* (2004) give such an implicit model based on a transportation problem to assign breaks to shifts. They show that the LP relaxation of their model is equivalent to the LP relaxations of two other classical implicit formulations, namely those proposed in Aykin (1996) and in Bechtold and Jacobs (1990). Since Dantzig's set covering model has the same integrality gap as Aykin's model, the LP relaxations of these four models are equivalent. Rekik *et al.* (2010) propose extensions to allow more flexibility in the definition of breaks. However, to this date, we are not aware of any implicit formulation that can accurately represent *multi-activity* shift scheduling problems.

An alternative to existing explicit and implicit models is to use formal languages to model work regulations. Côté *et al.* (2007) propose an IP model based on a regular language, represented by a finite deterministic automaton, to formulate the constraints defining a shift, and to represent all feasible shifts using a network flow formulation. Côté *et al.* (2009) extend these results by using context-free grammars in modeling shift scheduling problems. From a grammar describing work regulations, they generate an IP model based on assignment variables y_{ije}, that describe all feasible shifts for each employee e. Since the number of employees is bounded from below by $\max_{i \in I} \sum_{j \in J} b_{ij}$, this model generates a large number of variables. Moreover, in the case where many employees are alike, this model has symmetry issues.

Multi-activity shift scheduling problems. With the use of formal languages, many constraints in the planning of shifts can be considered. In particular, these modeling methods can deal with contexts where multiple work-activities can be performed during the same shift, each activity having its own labor requirements. Compact models for multi-activity problems are not common in the literature. Among the few papers addressing this topic, Loucks and Jacobs (1991) and Ritzman *et al.* (1976) model the tour scheduling problem (shift scheduling over one week) with Boolean assignment variables specifying the number of employees assigned to a given task at any given time. Since such modeling approaches yield very large IP formulations, both

papers propose heuristic methods to construct and improve the solutions. Moreover, they do not place breaks or meals during the shifts, nor do they handle regulations concerning the transition between activities. Approaches using column generation were suggested in Bouchard (2004); Vatri (2001) and Demassey *et al.* (2006). The first two propose approaches to schedule air traffic controllers. While Vatri (2001) uses a heuristic method to build the schedule without taking into account break placement, Bouchard (2004) extends his work to include break placement and solves the problem with a heuristic column generation approach. Demassey *et al.* (2006) propose a column generation procedure based on constraint programming, that solves efficiently the LP relaxation of the problem stated in Section 5.5.2 of this paper for up to 10 work-activities. However, they report that branching to find integer solutions is difficult and succeeds only for the smallest instances. More recently, Lequy *et al.* (2009) address another type of multi-activity shift scheduling problem where shifts and breaks are fixed a priori for each employee and where work-activities must then be assigned to shifts. Each employee has a set of skills that restricts the set of activities he can perform. Lequy *et al.* (2009) present three integer programming models and show good computational results with a heuristic column generation method embedded within a rolling horizon procedure.

In the following, we study some basic properties of grammars and show how they can be used in the context of shift scheduling problems.

5.2.2 Grammars

A context-free grammar defines a language over a given alphabet by means of a set of rules called *productions*. A production is a rule that specifies a substitution of symbols. These symbols are of two types: the *terminal symbols* are letters of the alphabet, generally represented by lower case letters, and the *non-terminal symbols* designate a subsequence that could be rewritten using the associated productions, generally represented by upper case letters. More formally, a production is represented as follows: $\alpha \rightarrow \beta$, where α is a non-terminal symbol and β is a sequence of terminal and/or non-terminal symbols. The productions of a grammar can be used recursively to generate new symbol sequences until only terminal symbols are part of the sequence. A sequence of terminal symbols is called a *word*.

Definition. A *context-free grammar* G is characterized by a tuple (Σ, N, P, S) where:

- Σ is an alphabet;
- N is a set of non-terminal symbols;
- P is a set of productions;
- S is the starting non-terminal.

A word, or sequence of letters from alphabet Σ, is *recognized* by a grammar G if it can be generated by successive applications of productions from G, starting with non-terminal S.

In the following, we will use the term grammar to refer to a context-free grammar and we will assume that, except when specified otherwise, all grammars are in Chomsky normal form, meaning that all productions are of the form $X \rightarrow \beta$ where $X \in N$ and $\beta \in (N \times N) \cup \Sigma$. Note that this assumption is not restrictive since any context-free grammar can be converted to Chomsky normal form; see Hopcroft *et al.* (2001) for more information on formal languages.

Example 4 *The following grammar G defines all feasible shifts for a simple shift scheduling problem. A shift must have a duration equal to the planning horizon and contain one break of one period anywhere during the shift except at the first or the last period. Work and break periods are respectively represented by letters w and b.*
$G = (\Sigma = (w, b), N = (S, X, W, B), P, S)$, where P is:
$S \rightarrow XW, \quad X \rightarrow WB, \quad W \rightarrow WW \mid w, \quad B \rightarrow b,$
*where the symbol $\mid$ specifies a choice of production. The shifts wbw, $wwwwwbw$ and $wbww$, among others, are recognized by G. $wbwb$ is not recognized by G. Word $wwbw$ is obtained by the derivation shown in Table 5.1, where **P** is the production used and **CS** is the current sequence, obtained from the previous sequence by applying the production on the left side.*

A common way to illustrate the derivation of a word from a grammar is to use a tree, called *parse tree*, where the root node is the starting non-terminal S, the interior nodes are non-terminals and leaves are letters of the alphabet. A production $X \rightarrow YZ$ is represented by nodes Y and Z as left and right children of node X, while $X \rightarrow a$ is represented by node X and a unique child, leaf a. When listed from left to right, the leaves form a word recognized by the grammar. Figure 5.1 shows the two parse trees induced by grammar G from Example 4 on words of length four ($wwbw$ and $wbww$).

TABLE 5.1 Derivation of word $wwbw$ from grammar G of Example 4

P	**CS**
–	S
$S \to XW$	XW
$X \to WB$	WBW
$W \to WW$	$WWBW$
$W \to w$	$wWBW$
$W \to w$	$wwBW$
$B \to b$	$wwbW$
$W \to w$	$wwbw$

A parse tree representing a word ω of length n has the following properties:

- An interior node and its children represent a production in P.
- A leaf is associated with a position $i \in \{1, \ldots, n\}$ in ω and represents the letter from Σ taking place at position i.
- Any interior node is the root of a tree inducing a subsequence of ω, starting at position $i \in \{1, \ldots, n\}$ with length $l \in \{1, \ldots, n - i + 1\}$.

Using these observations, the next developments characterize a graph embedding all parse trees associated to words of a given length.

The DAG Γ. In the following, we describe a *directed acyclic graph* (DAG) Γ that encapsulates all parse trees associated to words of a given length n recognized by a grammar $G = (\Sigma, N, P, S)$. The DAG Γ has an and-or structure containing two types of nodes: nodes O (the or-nodes) represent non-terminals from N and letters from Σ, and nodes A (the and-nodes) represent productions from P. Each node is characterized by its symbol (non-terminal, letter, or production) and the position and length of the subsequence it generates. We define O_{il}^{π} the node associated with non-terminal or letter π that generates a subsequence at position i of length l. Note that if $\pi \in \Sigma$, the node is a leaf and l is equal to one. Also, Γ has a root node described by O_{1n}^{S}. Likewise, $A_{il}^{\Pi,t}$ is the t^{th} node representing production Π generating a subsequence from position i of length l. There are as many $A_{il}^{\Pi,t}$ nodes as there are ways of using Π to generate a sequence of length l from position i. We will refer to this set as the (potentially empty) set $A(\Pi, i, l)$.

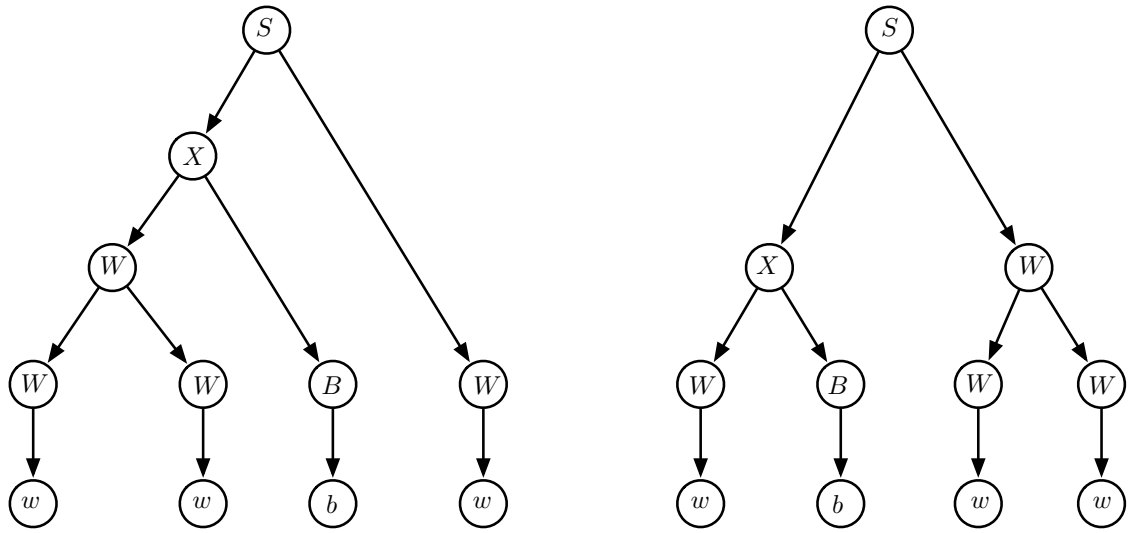

FIGURE 5.1 Parse trees for grammar G from Example 4 on words of length 4.

The DAG Γ is built in such a way that a path from one node to any other node alternates between or-nodes O and and-nodes A. More precisely, the DAG Γ has the following properties:

- Children of an or-node O_{il}^{π} with $l > 1$, denoted $ch(O_{il}^{\pi})$, are all and-nodes $A_{il}^{\Pi,t}$ such that $\Pi : \pi \to \beta$, $\beta \in (N \times N) \cup \Sigma$ and $t \in A(\Pi, i, l)$.
- Each or-node O_{i1}^{π} where π is a non-terminal has only one child: $ch(O_{i1}^{\pi}) = A_{i1}^{\Pi,1}$ such that $\Pi : \pi \to a$, where $a \in \Sigma$.
- Parents of an or-node O_{il}^{π} where $\pi \neq S$ is a non-terminal, denoted $par(O_{il}^{\pi})$, are and-nodes of the form $A_{jm}^{\Pi,t}$ such that $\Pi : X \to \pi Z$ or $\Pi : X \to Y\pi$, where $j \leq i$ and $m \geq l$.
- Each and-node $A_{il}^{\Pi,t}$ with $l > 1$ such that $\Pi : X \to YZ$ has exactly two children: O_{ik}^{Y} and $O_{i+k,l-k-1}^{Z}$, where $k < l - 1$.
- Each and-node $A_{i1}^{\Pi,1}$ such that $\Pi : X \to a$, where $a \in \Sigma$, has only one child: O_{i1}^{a}.
- Each and-node $A_{il}^{\Pi,t}$ has only one parent: O_{il}^{π} such that $\Pi : \pi \to \beta$, $\beta \in (N \times N) \cup \Sigma$, if $l > 1$, and $\Pi : \pi \to a$, $a \in \Sigma$, if $l = 1$.

Figure 5.2 presents the DAG Γ associated with grammar G from Example 4 on a word of length 4. It is easy to verify the above properties on this DAG.

To derive any parse tree from Γ, we start at the root O_{1n}^{S}. We visit an or-node O_{il}^{π} by selecting exactly one child, which is necessarily an and-node. We visit an and-node $A_{il}^{\Pi,t}$ by choosing all its children (exactly two if $l > 1$, one otherwise). By traversing Γ in this way until the only remaining unvisited nodes are leaves, we obtain a parse tree associated to the word defined by the leaves. Conversely, starting from a given word ω, we can traverse Γ backwards in a straightforward way to derive the parse tree associated to ω. In practice, Γ is built by a procedure suggested in Quimper and Walsh (2007) inspired by an algorithm from Cooke, Younger, and Kasami (see Hopcroft *et al.* (2001)).

Grammar-based IP model. Using the structure of the DAG Γ, Côté *et al.* (2009) present a system of linear equations in 0-1 variables that allow the identification of any word recognized by a given grammar G. To each node O_{il}^{π} and $A_{il}^{\Pi,t}$ in Γ are associated 0-1 variables u_{il}^{π} and $v_{il}^{\Pi,t}$, respectively. If we denote by L, the set of leaves in Γ, these equations are as follows:

$$u_{il}^{\pi} = \sum_{A_{il'}^{\pi,t} \in ch(O_{il}^{\pi})} v_{il}^{\Pi,t}, \qquad \forall O_{il}^{\pi} \in O \setminus L, \tag{5.6}$$

$$u_{il}^{\pi} = \sum_{A_{il'}^{\Pi,t} \in par(O_{il}^{\Pi})} v_{il}^{\Pi,t}, \qquad \forall O_{il}^{\pi} \in O \setminus O_{1n}^{S}, \tag{5.7}$$

$$u_{il}^{\pi} \in \{0,1\}, \qquad \forall O_{il}^{\pi} \in O, \tag{5.8}$$

$$v_{il}^{\Pi,t} \in \{0,1\}, \qquad \forall A_{il}^{\Pi,t} \in A. \tag{5.9}$$

Constraints (5.6) ensure that if variable u_{il}^{π} is equal to one, exactly one of the variables associated with its children must be equal to one. Similarly, constraints (5.7) ensure that if variable u_{il}^{π} is equal to one, exactly one of the variables associated with its parents must be equal to one. Consequently, when we set $u_{1n}^{S} = 1$, if this system of equations has a solution, then, in any solution, the variables equal to one form a parse tree associated to a word of length n recognized by G. Conversely, let ω be a word of length n on alphabet Σ. If we set to one the u_{i1}^{j} variables that form ω when the letters j are listed from left to right, then, if this system of equations has a solution, ω is recognized by G and the variables of the solution set to one form a

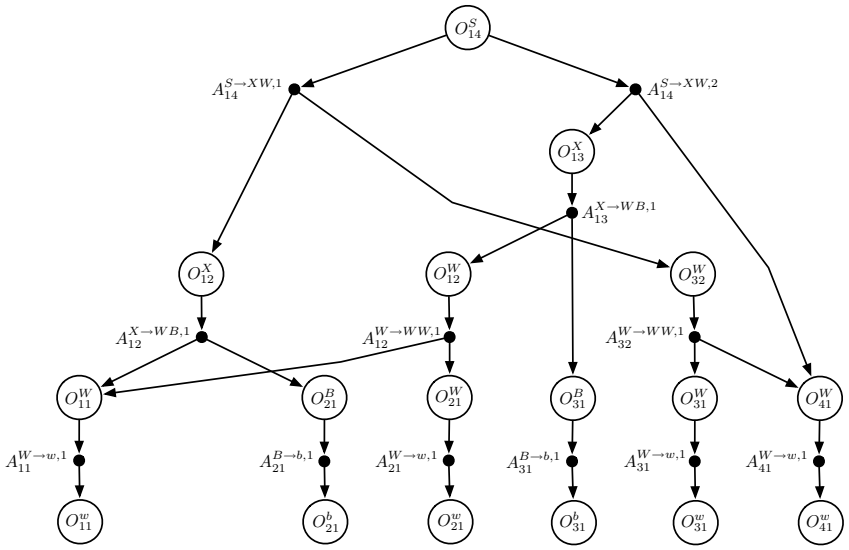

FIGURE 5.2 DAG Γ for grammar from Example 4 on a word of length 4.

parse tree associated to word ω.

We can rewrite equations (5.6) and (5.7) as follows:

$$u_{1n}^{S} = \sum_{A_{1n}^{\Pi,t} \in ch(O_{1n}^{S})} v_{1n}^{\Pi,t}, \tag{5.10}$$

$$\sum_{A_{il}^{\Pi,t} \in par(O_{il}^{\pi})} v_{il}^{\Pi,t} = \sum_{A_{il}^{\Pi,t} \in ch(O_{il}^{\pi})} v_{il}^{\Pi,t}, \qquad \forall O_{il}^{\pi} \in O \setminus \left\{ L \cup \{O_{1n}^{S}\} \right\}, \tag{5.11}$$

$$u_{i1}^{j} = \sum_{A_{i1}^{\Pi,t} \in par(O_{i1}^{j})} v_{i1}^{\Pi,t}, \qquad \forall O_{i1}^{j} \in L. \tag{5.12}$$

This system of equations presents a structure similar to network flow conservation equations, but the two systems are different. Indeed, a solution to equations (5.10)-(5.12) does not specify a path in a network, but rather a tree in the DAG Γ, since any variable associated with an and-node which is equal to one in a solution will also have its two children with variables equal to one. Hence, (5.10)-(5.12) are not flow conservation equations. Further, if we represent system (5.10)-(5.12) in matrix notation, we can easily show that the corresponding matrix is not totally unimodular, contrary to the incidence matrix of a network, which is used to represent flow conservation equations. In spite of this, Pesant *et al.* (2009) have shown (see Section 5.4) that the

polyhedron defined by (5.10)-(5.12) is integral, like the polyhedron defined by flow conservation equations.

5.3 Grammar-Based Model for Shift Scheduling

As explained in Côté *et al.* (2009), the system of equations (5.10)-(5.12) can be used in the context of shift scheduling problems, where the constraints defining any feasible shift are represented by a grammar G, i.e., each word ω recognized by grammar G corresponds to a feasible shift $s \in \Omega$. In this context, the number of periods $|I|$ corresponds to n, the length of any given word recognized by G, while the set of activities J corresponds to Σ, the letters of the alphabet.

Côté *et al.* (2009) describe an IP model based on this correspondance, using assignment variables y_{ije}, that describe all feasible shifts for each employee e. But, as explained in Section 5.2.1, when employees are similar, this model exhibits a lot of symmetry, which makes it impractical to solve large-scale instances. Assuming all employees can be assigned to the same shifts, we introduce here a new grammar-based IP model, that will not suffer from the same performance issues.

In equations (5.10)-(5.12), each variable is binary and specifies whether or not its corresponding node is part of the parse tree selected to generate a word. In the new model, each variable is a nonnegative integer that specifies how many parse trees the associated node is part of. Since we minimize an objective function with nonnegative costs, the integer variables do not need to be bounded from above.

As in the Introduction, let y_{ij} denote the number of employees assigned to activity $j \in J$ at period $i \in I$. We can replace the leaf variables u^j_{i1} by the variables y_{ij}. Model Q presented in the Introduction can now be explicitly stated as follows:

$$f(Q) = \min \sum_{i \in I} \sum_{j \in J} c_{ij} y_{ij}$$

$$y_{ij} \geq b_{ij}, \qquad \forall i \in I, j \in J, \tag{5.13}$$

$$u_{1n}^{S} = \sum_{A_{1n}^{\Pi,t} \in ch(O_{1n}^{S})} v_{1n}^{\Pi,t}, \tag{5.14}$$

$$\sum_{A_{il}^{\Pi,t} \in par(O_{il}^{\pi})} v_{il}^{\Pi,t} = \sum_{A_{il}^{\Pi,t} \in ch(O_{il}^{\pi})} v_{il}^{\Pi,t}, \qquad \forall O_{il}^{\pi} \in O \setminus \left\{ L \cup \{O_{1n}^{S}\} \right\}, \tag{5.15}$$

$$y_{ij} = \sum_{A_{i1}^{\Pi,t} \in par(O_{i1}^{j})} v_{i1}^{\Pi,t}, \qquad \forall i \in I, j \in J, \tag{5.16}$$

$$u_{1n}^{S} \geq 0 \text{ and integer}, \tag{5.17}$$

$$v_{il}^{\Pi,t} \geq 0 \text{ and integer}, \qquad \forall A_{il}^{\Pi,t} \in A, \tag{5.18}$$

$$y_{ij} \geq 0 \text{ and integer}, \qquad \forall i \in I, j \in J. \tag{5.19}$$

Once Q is solved, an implicit solution is obtained. To find the individual schedules from this solution, we traverse the DAG Γ from the root to the leaves visiting the nodes with value greater than zero. Each time a node is evaluated, its value is decreased by one. When a leaf node is reached, its value is inserted to the current schedule at the right position (see Appendix A for the detailed algorithm). Model Q ensures that u_{1n}^{S} words recognized by grammar G can be extracted from the implicit solution. In the context of shift scheduling, variable $u_{1n}^{S} = k$ thus represents the total number of employees needed to perform all required shifts. The complexity of the algorithm used to extract the explicit set of shifts from the optimal implicit solution is $O(kn^3|G|)$ where n is the sequence length and $|G|$ is the number of productions in grammar G. In practice, the running time to perform this algorithm is negligible compared to the time necessary to solve model Q.

5.4 Theoretical Properties of Grammar-Based Models

In this section, we study the polyhedral properties of the implicit grammar-based model Q and compare it with other models from the literature. First, using the

notation from the Introduction, we denote by H the polyhedron defined by equations (5.14)-(5.16) along with nonnegativity constraints on all variables. Our first result states that this polyhedron is integral, thus extending the result derived in Pesant *et al.* (2009) for a similar polyhedron defined over 0-1 variables.

Theorem 3 *H is an integral polyhedron.*

Proof: To simplify the notation, we denote H using matrix notation as follows: $H = \{z \geq 0 | Mz = b\}$. Now, let d be any arbitrary costs associated with variables z. The result will follow if we can prove that there always exists an integer optimal solution to the linear program: $\min\{dz | z \in H\}$. For this, it suffices to construct an integer point z^I in H that satisfies the complementary slackness conditions: $(\lambda^* M - d)z^I = 0$, where λ^* is an optimal solution to the dual $\max\{\lambda b | \lambda M \leq d\}$.
Let z^* be an optimal solution to the linear program and let λ^* be the corresponding dual solution. We assume that $m = \lceil u^S_{1n} \rceil > 0$ (otherwise, if $m = 0$, $z^I = 0$ is an integer point in H satisfying the complementary slackness conditions). Our objective is to construct an integer solution z^I such that for every row k for which $\lambda^* M_k < d_k$, we have $z^I_k = 0$. This condition can be easily maintained by enforcing that $z^I_k = 0$ whenever $z^*_k = 0$.
First, set $u^S_{1n} = m$ in z^I. By definition of H, since $u^S_{1n} > 0$ in z^*, there exists at least one variable corresponding to a child of root-node O^S_{1n} that has a value greater than 0 in z^*, say z^*_k corresponding to node $A^{\Pi,t}_{1n}$, with $\Pi : X \rightarrow YZ$. We continue our construction by fixing $z^I_k = m$. From constraints (5.15), since $z^*_k > 0$, the two children of $A^{\Pi,t}_{1n}$, say O^X_{1k} and $O^Y_{k+1,n-k}$ have at least one child each with a corresponding value greater than 0, say $z^*_{k_1}$ and $z^*_{k_2}$. We then fix $z^I_{k_1} = m$ and $z^I_{k_2} = m$. We continue this process, following the children of the nodes and setting them to m in z^I, until we reach the leaves of the DAG Γ.
The variables set to m in z^I form a tree in the DAG Γ that satisfies constraints $Mz = b$ by construction. Furthermore, since we only used variables that were already set to a value greater than 0 in the LP optimal solution z^*, we know that z^I satisfies the complementary slackness conditions with respect to λ^*. Therefore, there always exists an optimal integer solution to the linear program $\min\{dz | z \in H\}$, with arbitrary d, i.e., polyhedron H is integral. ■

From this result, we observe that integrality constraints (5.17) and (5.18) are redundant in model Q. However, in practice, we found that leaving these constraints

in the formulation helps the IP solver to further presolve the model and, overall, speeds up the solution process. Consequently, the experimentions in Section 5.5 were performed by leaving the integrality constraints in the models.

The next theorem uses the previous result to establish the equivalence between the LP relaxations of models Q and D, the Dantzig set covering formulation presented in the Introduction.

Theorem 4 *Q and D have equivalent LP relaxations.*

Proof: The proof is direct using Lagrangean duality arguments. Let $\gamma_{ij} \geq 0$ denote Lagrangean multipliers associated with the requirement constraints, (5.1) in model D and (5.3) in model Q. Also, let $f_{LP}(M)$ denote the optimal objective value of the LP relaxation of a model M. We then have:

$$
\begin{aligned}
& f_{LP}(Q) \\
& = \max_{\gamma \geq 0} \left\{ \sum_{i \in I} \sum_{j \in J} \gamma_{ij} b_{ij} + \min \left\{ \sum_{i \in I} \sum_{j \in J} (c_{ij} - \gamma_{ij}) y_{ij} \,|\, (y, v) \in H \right\} \right\} \\
& = \max_{\gamma \geq 0} \left\{ \sum_{i \in I} \sum_{j \in J} \gamma_{ij} b_{ij} + \min \left\{ \sum_{i \in I} \sum_{j \in J} (c_{ij} - \gamma_{ij}) y_{ij} \,|\, (y, v) \in H, (y, v) \text{ integer} \right\} \right\} \\
& = \max_{\gamma \geq 0} \left\{ \sum_{i \in I} \sum_{j \in J} \gamma_{ij} b_{ij} + \min \left\{ \sum_{i \in I} \sum_{j \in J} (c_{ij} - \gamma_{ij}) (\sum_{s \in \Omega} \delta_{ijs} x_s) \,|\, x_s \geq 0 \text{ integer} \right\} \right\} \\
& = \max_{\gamma \geq 0} \left\{ \sum_{i \in I} \sum_{j \in J} \gamma_{ij} b_{ij} + \min \left\{ \sum_{s \in \Omega} (c_s - \sum_{i \in I} \sum_{j \in J} \gamma_{ij} \delta_{ijs}) x_s \,|\, x_s \geq 0 \text{ integer} \right\} \right\} \\
& = \max_{\gamma \geq 0} \left\{ \sum_{i \in I} \sum_{j \in J} \gamma_{ij} b_{ij} + \min \left\{ \sum_{s \in \Omega} (c_s - \sum_{i \in I} \sum_{j \in J} \gamma_{ij} \delta_{ijs}) x_s \,|\, x_s \geq 0 \right\} \right\} \\
& = f_{LP}(D).
\end{aligned}
$$

■

Since Dantzig's set covering model yields the same integrality gap as the models suggested in Aykin (1996); Bechtold and Jacobs (1990) and Rekik *et al.* (2004) (see Section 5.2), Theorem 4 implies that the LP relaxation of P is also equivalent to the LP relaxations of these other implicit models. Note however that, to the best of our knowledge, these models cannot be extended to the multi-activity case.

5.5 Computational Experiments

The objective of our computational experiments is to evaluate the efficiency of our new implicit grammar-based model, when processed by a state-of-the-art IP solver. For this purpose, we will first compare model Q to the shift-based explicit model D and to another implicit model, due to Aykin (1996), under the same conditions. In order to compare ourselves to these other modeling approaches, we will first use instances from the literature, all having one work-activity. Then, we will present computational results on large-scale instances with multiple work-activities and compare our approach to employee-based explicit models tested by Côté *et al.* (2009) on the same instances.

5.5.1 Shift Scheduling with Multiple Rest Breaks, Meal Breaks, and Break Windows

In this section, we compare our model with a state-of-the-art implicit model, proposed in Aykin (1996), and to the shift-based explicit model D on a large set of shift scheduling instances used in Mehrotra *et al.* (2000) from shift specifications and labor requirements reported in Aykin (1996); Henderson and Berry (1976); Segal (1974) and Thompson (1995a). The problems differ from one another in the labor requirements, the set of allowed shifts, the planning horizon, the number of breaks, the break windows, the cost structures, and whether the problem is cyclic or not. We refer to Mehrotra *et al.* (2000) for details on shift generation rules. Here, we present a general description of the three classes of problems studied.

Thompson95a Set. Thompson (1995a) presents two sets of non-cyclic problems. The first set contains problems on 15-h demand patterns. Shifts either allow one break or none depending on their length. The second set contains problems on 20-h demand patterns. Shifts allow one break of one hour. The planning horizons are divided into periods of 15 minutes and shifts can start at any period that allows them to finish within the planning horizon. The break windows depend on the duration of the shifts and the costs are proportional to the number of work hours in a shift.

Aykin Set. Aykin (1996) presents a set of cyclic problems, with shifts containing exactly three breaks and differing only in the length of the break windows. The

planning horizon is 24-h divided into periods of 15 minutes. All shifts have the same length and must start on the hour or the half-hour. The cyclic case is handled in the same way in the three modeling approaches. The planning horizon is extended to allow shifts to start at any time in the original planning horizon.

Mehrotra Set. Mehrotra *et al.* (2000) use the same shift generation rules as Aykin, but allow the shifts to have different durations and to start at any period. The problems were tested as cyclic problems using the same labor requirements as in the Aykin Set.

Definition of the Grammars. The following presents the grammars used for each set of instances. For the sake of clarity, the grammars are not stated in Chomsky normal form. In the sets of productions P, $\rightarrow_{[min,max]}$ restricts the subsequences generated with a given production to have a length between min and max periods.

Grammar for Thompson95a Set. Let Φ be the set of feasible shift types. Let bws_l and bwe_l be the break window starting and ending periods for shift type $l \in \Phi$. Let sl_l and bl_l be the shift and break lengths for shift type $l \in \Phi$.
Then $G = (\Sigma = (w, b, r), N = (S, W, R, A_l, B_l, M_l \ \forall l \in \Phi), P, S)$,
where w is a period of work, b is a break period and r is a rest period. P is defined as follows:

$$
\begin{array}{lll}
S \rightarrow RA_lR \mid A_lR \mid RA_l & \forall l \in \Phi, & W \rightarrow Ww \mid w, \\
A_l \rightarrow_{[sl_l, sl_l]} M_lW & \forall l \in \Phi, & R \rightarrow Rr \mid r, \\
M_l \rightarrow_{[bws_l + bl_l, bwe_l + bl_l]} WB_l & \forall l \in \Phi, & \\
B_l \rightarrow b^{bl_l}, & \forall l \in \Phi. &
\end{array}
$$

Grammar for Aykin and Mehrotra Sets. Let Φ be the set of feasible shift types. Note that in the Aykin Set Φ contains only one element, since all shifts have the same length. Let bws_l^n and bwe_l^n be the break window starting and ending periods for shift type $l \in \Phi$ and break $n \in \{1, 2, 3\}$. Let sl_l and bl_l^n be the length of the shift and of the breaks $n \in \{1, 2, 3\}$, respectively, for shift type $l \in \Phi$.
Then $G = (\Sigma = (w, b, r), N = (S, W, R, B_l^n \ \forall n \in \{1, 2, 3\}, A_l, M_l^A, M_l^B, M_l^C \ \forall l \in \Phi), P, S)$,
where w is a period of work, b is a break period and r is a rest period. P is defined

as follows:

$$
\begin{array}{lll}
S \rightarrow RA_lR \mid A_lR \mid RA_l & \forall l \in \Phi, & B_l^n \rightarrow b^{bl_l^n} \quad \forall l \in \Phi, n \in \{1,2,3\}, \\
A_l \rightarrow_{[sl_l, sl_l]} M_l^A W & \forall l \in \Phi, & W \rightarrow Ww \mid w, \\
M_l^A \rightarrow_{[bws_l^3+bl_l^3, bwe_l^3+bl_l^3]} M_l^B W B_l^3 & \forall l \in \Phi, & R \rightarrow Rr \mid r, \\
M_l^B \rightarrow_{[bws_l^2+bl_l^2, bwe_l^2+bl_l^2]} M_l^C W B_l^2 & \forall l \in \Phi, & \\
M_l^C \rightarrow_{[bws_l^1+bl_l^1, bwe_l^1+bl_l^1]} W B_l^1 & \forall l \in \Phi. &
\end{array}
$$

Results. We compare our model on these instances to the shift-based explicit model D derived from Dantzig (1954) and to the implicit model from Aykin (1996). We generated the three IP models and solved them with CPLEX 10.1.1 with the default parameters. As in Mehrotra *et al.* (2000), we gave a four-minute time limit to find an optimal solution. Experiments were run on a 2.3GHz AMD Opteron with 3GB of memory.

Table 5.2 shows the number of instances solved to optimality within the four-minute time limit by the three models on each set. The numbers in parentheses are the number of available instances in each set.

For the instances solved to optimality by all three models, Table 5.3 presents a summary of the model sizes and solution statistics. $|C|$, $|V|$ and $|NZ|$ are the number of constraints, variables and non-zeroes in the models. Nit, $Nnodes$ and $Time(s)$ give the average values of the number of simplex iterations, the number of nodes and the time (in seconds) needed to solve the instances to optimality. $Gap(\%)$ is the average IP gap for the instances that were not solved to optimality by at least one of the three models, i.e., $Gap = 100(Z_{IP} - Z_{LP})/Z_{IP}$ where Z_{IP} and Z_{LP} are, respectively, the best upper and lower bounds after the time limit has been reached.

The results on the Thompson95a Set show that our model can lead to better solution times in comparison with the two other formulations. Indeed, with the

TABLE 5.2 Number of instances solved to optimality within the four- minute time limit

Models/Sets	Thom. 15-h (21)	Thom. 20-h (80)	Aykin (16)	Mehrotra (16)
Implicit Grammar	21	79	16	16
Aykin	21	66	16	16
Dantzig	21	63	16	16

TABLE 5.3 Summary for the instances solved to optimality

Model	$\|C\|$	$\|V\|$	$\|NZ\|$	Nit	$Nnodes$	$Time(s)$	$Gap(\%)$
			Thompson95a set for 15-h demand curves				
Implicit Grammar	6418	9686	28481	400	591	0.46	–
Aykin	421	3673	27521	3467	13	0.93	–
Dantzig	60	3312	85977	185	10	0.34	–
			Thompson95a set for 20-h demand curves				
Implicit Grammar	13732	22118	65582	1338	11	4.04	0.004
Aykin	838	9208	67912	5866	130	4.89	0.082
Dantzig	80	8450	229215	751	35	3.82	0.140
			Aykin set				
Implicit Grammar	5703	36184	106902	3421	17	3.59	–
Aykin	274	697	3396	271	2	0.07	–
Dantzig	130	4681	149760	860	36	0.77	–
			Mehrotra et al. set				
Implicit Grammar	11201	16488	47683	15629	15	14.50	–
Aykin	1572	6961	33955	2006	2	1.06	–
Dantzig	132	46801	1497113	659	5	8.90	–

Implicit Grammar model, we manage to solve to optimality 13 instances more than Aykin and 16 more than Dantzig. However, for instances in the Aykin and the Mehrotra Sets, our model appears less effective, although it finds the optimal solution for all instances within the time limit, as the two other models do.

Mehrotra *et al.* (2000) present a branch-and-price approach involving specialized branching rules for solving Dantzig set covering formulation. They compare their method with Aykin's model solved with CPLEX 4.0 on the same instances stated above. The results show that their method is generally superior. Since CPLEX has evolved considerably since these experiments were performed, it is difficult to deduce from these results a fair comparison between their approach and our model solved with CPLEX 10.1.1.

5.5.2 Shift Scheduling with Multiple Rest and Meal Breaks, and Multiple Work Activities

This section presents a shift scheduling problem for a retail store, allowing up to ten different work activities. We present the specifications of the problem and first compare our model with Dantzig's model and an extension of Aykin's model suggested in Rekik *et al.* (2010), allowing to model work-stretch duration restrictions for instances with one work-activity. Then, we report solution times from Côté *et al.* (2009) on the instances with up to two work-activities and compare them with the results from our model.

Problem Definition

1. The planning horizon is 24 hours divided into 96 periods of 15 minutes.
2. A shift may start at any period of the day allowing enough time to complete its duration during the planning horizon.
3. A shift must cover between 3 hours and 8 hours of work activities.
4. If a shift covers at least 6 hours of work activities, it must have two 15-minute breaks and a lunch break of 1 hour.
5. If a shift covers less than 6 hours of work activities, it must have one 15-minute break, but no lunch.

6. If performed, the duration of a work-activity is at least 1 hour (4 consecutive periods).
7. A break (or lunch) is necessary between two different work activities.
8. Work activities must be inserted between breaks, lunch and rest stretches.
9. For each period of the planning horizon, labor requirements for every work-activity are available.
10. Overcovering and undercovering are allowed. Costs are associated with overcovering and undercovering the requirements of a work-activity at a given period.
11. The cost of a shift is the sum over every period of the costs of all work-activities performed in the shift.

Definition of the Grammar. The following presents the grammar used for this problem. For the sake of clarity, the grammar is not stated in Chomsky normal form.
$G = (\Sigma = (a_j \ \forall j \in A, b, l, r), N = (S, F, P, W, A_j \ \forall j \in A, B, L, R), P, S)$,
where A is the set of work-activities, a_j is a period of work on activity $j \in A$, b is a break period, l is a lunch period and r is a rest period. In P, $\rightarrow_{[min,max]}$ restricts the subsequences generated with a given production to have a length between min and max periods. P is defined as follows:

$$
\begin{array}{ll}
S \rightarrow RFR \mid FR \mid RF \mid RPR \mid PR \mid RP, & B \rightarrow b, \\
F \rightarrow_{[30,38]} WBWLWBW \mid WLWBWBW \mid WBWBWLW, & L \rightarrow llll, \\
P \rightarrow_{[13,24]} WBW, & R \rightarrow Rr \mid r, \\
W \rightarrow_{[4,\infty)} A_j \quad \forall j \in A, & \\
A_j \rightarrow A_j a_j \mid a_j \quad \forall j \in A. &
\end{array}
$$

Results. To compare the different models for this problem, we generated the IP models representing these rules and solved them with CPLEX 10.1.1 with the default parameters. We gave a one-hour time limit to find an optimal solution. Experiments were run on a 3.20 GHz Pentium 4.

First, we compare Dantzig's model and the extension of Aykin's model suggested in Rekik *et al.* (2010), called the Aykin/Rekik model, to our model on ten instances with one work-activity, which differ only in their labor requirements. Table 5.4 presents the results. $|C|$, $|V|$ and $|NZ|$ are the number of constraints, variables and non-zeroes in the models. Note that the differences in the number of variables between the instances for the same model come from the slack variables introduced to

allow overcovering and undercovering of requirements constraints. For periods where no employees are required, we suppose that the retail store is closed and that no work should be scheduled. Nit, $Nnodes$ and $Time(s)$ give the number of simplex iterations, nodes and the solution times (in seconds) for the instances solved to optimality within the time limit; otherwise the sign ">" is used.

The comparison between the three models shows that Dantzig's model tends to be less competitive on problems with a large number of shifts. In the one-activity case, solving the entire model with an IP solver is still manageable, but both the Implicit Grammar models and Aykin/Rekik models are solved more rapidly (except for instance 9, for which Dantzig's model performs slightly better than the Aykin/Rekik model). Our model succeeds in proving optimality for 9 out of 10 instances, as does the Aykin/Rekik model, but does so in less time for 7 out of these 9 instances. Note also that the number of variables in the Implicit Grammar model is smaller than in the two other models.

Table 5.5 shows our results on the multi-activity instances. We ran experiments on our model on instances ranging from two to ten work activities. For each instance, we tested ten different labor requirements. Column $NbShifts$ gives the number of feasible shifts for each of the problems, which would be the number of variables needed by Dantzig's set covering model. $Nopt$ gives the number of instances solved to optimality within the one-hour time limit. $|C|$, $|V|$ and $Time(s)$ are the average number of constraints and variables, and solution times (in seconds) for the instances solved to optimality.

To our knowlegde, no other implicit formulations are capable of modeling multi-activity instances. To solve Dantzig's model on these problems, one must consider column generation methods, since the number of feasible shifts is very large. Demassey *et al.* (2006) present a column generation approach for these problems. However, their method does not succeed in finding optimal solutions, even for the single-activity instances. As for our modeling approach, the multi-activity problems can easily be handled with a few more productions than in the one-activity case, and results show that they can rapidly be solved on almost all available instances. Note that the growth in the number of constraints and variables when increasing the number of work activities is much slower than the increase in the number of feasible shifts.

TABLE 5.4 Model comparison on the one-activity problem

No	$\|C\|$	$\|V\|$	$\|NZ\|$	Nit	$Nnodes$	$Time(s)$
Implicit Grammar model						
1	16191	66621	198517	137	0	0.52
2	16191	66653	198549	67053	12	132.54
3	16191	66653	198549	762526	1312	838.79
4	16191	66637	198533	13245	37	9.05
5	16191	66629	198525	1154	0	0.69
6	16191	66629	198525	1006	0	0.60
7	16191	66637	198533	26715	95	11.60
8	16191	66653	198549	>	>	>
9	16191	63068	198525	12057	215	3.60
10	16191	66637	198533	1977	0	1.01
Aykin/Rekik model						
1	50007	78247	930056	604	0	2.31
2	50007	78279	930088	1776156	4313	2607.76
3	50007	78279	930088	231981	1061	415.75
4	50007	78263	930072	12382	140	12.23
5	50007	78255	930064	603	0	2.30
6	50007	78255	930064	825	0	2.33
7	50007	78263	930072	5742	74	7.60
8	50007	78279	930088	>	>	>
9	50007	78255	930064	64378	402	59.38
10	50007	78263	930072	2071	0	2.78
Dantzig model						
1	96	845176	24605722	23	0	45.42
2	96	845208	24605754	>	>	>
3	96	845208	24605754	561202	71033	1477.83
4	96	845192	24605738	70578	18270	128.75
5	96	845184	24605730	101	20	42.79
6	96	845184	24605730	23	0	46.08
7	96	845192	24605738	232	0	89.07
8	96	845208	24605754	>	>	>
9	96	845184	24605730	353	3266	46.05
10	96	845192	24605738	117	0	42.01

TABLE 5.5 Multi-activity problems with the Implicit Grammar model

$NbAct$	$NbShifts$	$\|C\|$	$\|V\|$	$Time(s)$	$Nopt(10)$
2	13404928	18068	69893	423.90	9
3	67752783	19945	73152	337.14	9
4	214010944	21822	76417	212.07	9
5	522350575	23699	79688	182.02	10
6	1082991744	25576	82961	179.51	10
7	2006203423	27453	86246	317.31	10
8	3422303488	29330	89492	274.66	10
9	5481658719	31207	92731	296.37	10
10	8354684800	33084	96026	518.02	10

Comparison with existing IP formulations based on formal languages. Table 5.6 presents the times reported by Côté *et al.* (2009) to solve the one and two work-activities instances with two employee-based explicit formulations, the *IP Regular* model, based on a finite automaton, and the *IP Grammar* model, based on a context-free grammar. In both cases, 0-1 assignment variables for each employee are used, instead of the general integer variables used in model Q. These experiments were run on a 2.4 GHz Dual AMD Opteron Processor 250 with 3 GB of RAM, using the MIP solver CPLEX 10.0 and a time limit of 3600 seconds. The table reports the time in seconds to obtain an integer solution with a relative IP gap smaller or equal than 1%. The symbol ">" represents an instance for which that gap could not be reached within the time limit. The *Implicit Grammar* column shows the time needed by the implicit grammar model to reach the 1% relative IP gap limit.

Table 5.6 illustrates that the two employee-based explicit formulations suffer from scalability issues as the number of work-activities grows. Observe that the implicit grammar model shows comparable solution times for both classes of instances. For the one-activity problem, the *IP Regular* model is much faster than the implicit grammar model on 3 out of 10 instances. However, for the instances with two work-activities, the implicit grammar model solves all instances more rapidly than the two other models, and succeeds in solving more instances within the time limit. It is interesting to note that, on almost all instances, the *IP Regular* model has better solution times than the explicit *IP Grammar* model, from which we built the implicit model presented in this paper.

TABLE 5.6 Comparison of solution times (seconds) between employee-based explicit formulations and the implicit grammar model on the one and two-activities instances to obtain a near-optimal solution ($Gap \leq 1\%$) in less than 3600 seconds

No	IP Regular	IP Grammar	Implicit Grammar
One-activity instances			
1	1.03	7.42	0.26
2	40.09	>	110.88
3	64.64	>	75.25
4	46.39	1850.38	2.75
5	14.03	322.57	0.48
6	3.28	130.21	0.34
7	5.99	1662.75	2.71
8	131.77	>	2642.12
9	16.14	1015.10	1.18
10	20.22	1313.28	0.80
Two-activities instances			
1	228.07	2826.40	1.27
2	2870.20	1952.58	4.12
3	1541.15	>	81.91
4	169.96	>	16.27
5	>	>	2.59
6	1288.56	>	51.16
7	29.94	>	0.60
8	>	325.08	36.20
9	>	>	>
10	1108.23	>	4.99

5.6 Conclusion

In this paper, we presented a new implicit IP model for solving multi-activity shift scheduling problems. This model differs significantly from the models proposed in the literature, as our modeling approach uses context-free grammars to represent the constraints defining feasible shifts. This model yields the same LP relaxation bound as the classical set covering model by Dantzig (1954) and other well-known implicit models in the literature, i.e., Aykin (1996); Bechtold and Jacobs (1990) and Rekik *et al.* (2004).

Our experiments showed that the solution times for our model are competitive with the solution times for Aykin's model on one-activity shift scheduling instances from the literature, and slightly superior to an extension of Aykin's model suggested in Rekik *et al.* (2004) on large-scale one-activity instances. We also showed that our model can be solved to optimality efficiently on instances with up to ten work activities. To the best of our knowledge, no other technique in the literature can solve multi-activity instances efficiently.

An interesting feature of our formulation is that the objective function allows many types of cost structures, contrary to classical implicit formulations. In model Q, costs are associated with activities and periods, but one can modify the objective function to have costs on productions ($\sum_{A_{il}^{\Pi,t} \in A} \text{cost}(v_{il}^{\Pi,t}) v_{il}^{\Pi,t}$), on the root-node (minimizing the root-node variable, would be equivalent to minimizing the number of employees needed), or even on subsequences. Indeed, $\sum_t v_{il}^{\Pi,t}$ corresponds to the number of words having a subsequence of length l starting in position i generated from production Π. A subsequence can be a shift or a part of a shift, such as a task. One could be interested in tracking a task corresponding to a consecutive assignment of work activities and do so by using the corresponding variables v from A. Therefore, the implicit grammar-based model can easily be extended to handle problems that require the simultaneous assignment of tasks and activities.

Acknowledgments

This work was supported by a grant from the Fonds québécois de recherche sur la nature et les technologies. We would like to thank Claude-Guy Quimper for his useful comments on our work. We would like to thank two anonymous referees whose constructive comments and questions helped us to improve our paper.

Chapitre 6

GRAMMAR-BASED COLUMN GENERATION FOR PERSONALIZED MULTI-ACTIVITY SHIFT SCHEDULING

Les méthodes implicites s'avèrent bien adaptées pour résoudre les problèmes de planification de quarts mono-activité et multi-activités anonymes. Dans plusieurs cas réels, la supposition voulant que tous les employés disponibles soient identiques ne peut être faite. En effet, dans la plupart des environnements de travail, les employés possèdent des caractéristiques individuelles les distinguant les uns des autres. Notamment, les employés peuvent posséder un ensemble de compétences leur permettant d'être affecté à un sous-ensemble des activités de travail à effectuer. Dans ces cas, les caractéristiques individuelles de chaque employé doivent être prises en compte lorsque les quarts devant leur être affectés sont sélectionnés.

Dans cet article, nous proposons une formulation et une méthode de résolution permettant d'aborder les problèmes de planification de quarts de travail personnalisés multi-activités. Le problème est d'abord modélisé par un modèle de recouvrement décomposé par employé, chaque employé possédant un ensemble de quarts réalisables différent. Ce type de formulation peut devenir rapidement impossible à résoudre directement en pratique, en particulier lorsque les nombres d'employés et de quarts sont très grands. Pour pallier cette difficulté, nous utilisons une méthode de génération de colonnes pour générer progressivement les quarts (colonnes) intéressants pour chaque employé, soit ceux permettant potentiellement d'améliorer l'objectif du modèle de recouvrement.

Pour générer les colonnes, nous proposons pour chaque employé un sous-problème basé sur les grammaires hors-contextes permettant de modéliser l'ensemble des quarts auxquels il peut être affecté. Donc, pour chaque employé disponible, une grammaire définit l'ensemble des quarts réalisables compte tenu de ses caractéristiques individuelles. À partir de cette grammaire, son graphe orienté acyclique est généré. Ce graphe est ensuite utilisé par un algorithme de programmation dynamique de manière à générer les colonnes permettant d'améliorer l'objectif du modèle principal lors de la résolution par la méthode de génération de colonnes.

La méthode de génération de colonnes seule permet a priori de résoudre la relaxation linéaire du modèle de recouvrement. Pour rechercher les solutions entières et ultimement, la solution entière optimale, nous proposons une stratégie de branchement s'incorporant à notre sous-problème de manière à pouvoir générer, à chaque noeud de l'arbre de branchement de nouvelles colonnes.

Finalement, nous étudions deux problèmes de planification de quarts de travail et nous comparons notre méthode à des approches proposées dans la littérature pour ces problèmes. L'utilisation de grammaires pour représenter les restrictions sur la composition des quarts nous permet de modéliser facilement ces deux problèmes très différents. De plus, les résultats expérimentaux démontrent que notre méthode peut efficacement résoudre ces problèmes à l'optimalité pour plusieurs instances.

L'article suivant a été soumis à la revue *INFORMS Journal on Computing* en juillet 2010.

Grammar-Based Column Generation for Personalized Multi-Activity Shift Scheduling

Marie-Claude Côté
Département de mathématiques et génie industriel, École Polytechnique de Montréal, PO Box 6079, succ. Centre-Ville, Montréal, H3C 3A7, Canada,
CIRRELT-Interuniversity Research Center on Enterprise Networks, Logistics and Transportation, Montréal, Canada, marie-claude.cote@polymtl.ca

Bernard Gendron
Département d'informatique et de recherche opérationnelle, Université de Montréal,Pavillon André-Aisenstadt, PO Box 6128, succ. Centre-Ville, Montréal, H3C 3J7, Canada,
CIRRELT-Interuniversity Research Center on Enterprise Networks, Logistics and Transportation, Montréal, Canada Bernard.Gendron@cirrelt.ca

Louis-Martin Rousseau
Département de mathématiques et génie industriel, École Polytechnique de Montréal, PO Box 6079, succ. Centre-Ville, Montréal, H3C 3A7, Canada,
CIRRELT-Interuniversity Research Center on Enterprise Networks, Logistics and Transportation, Montréal, Canada louis-martin.rousseau@polymtl.ca

Abstract

We present a branch-and-price algorithm to solve personalized multi-activity shift scheduling problems. The subproblems in the column generation method are formulated using grammars and solved with dynamic programming. The expressiveness of context-free grammars is exploited to easily model restrictions over shifts, allowing the branch-and-price algorithm to solve large-scale problem instances. We present computational experiments on two types of multi-activity shift scheduling problems and compare our approach with existing methods in the literature. These experiments show that our approach can solve efficiently large-scale instances and is flexible enough to model different classes of problems.

6.1 Introduction

We define the *personalized multi-activity shift scheduling problem* as follows. Given a planning horizon I divided into n periods of equal length, a set of work-activities J, a set of employees E, a set of feasible shifts Ω^e for each employee $e \in E$, and a number

of employees b_{ij} required at each period $i \in I$ for each work-activity $j \in J$, one must select for each employee $e \in E$ a feasible shift in Ω^e to cover the required number of employees at minimum cost, given that each feasible shift $s \in \Omega^e$ has an associated cost $c_s^e \geq 0$. We assume that the employees have different characteristics from one another, such as skills that allow them to perform only a subset of work-activities, or restrictions regarding their availability during some periods of the planning horizon. The set of feasible shifts for each employee is therefore determined by the skills, preferences and availability of each employee, but is also constrained by other rules arising from work regulation agreements and ergonomic considerations. This problem differs from the multi-activity shift scheduling problem studied in Côté *et al.* (2010), where all employees are assumed to be identical.

The classical set covering model, proposed in Dantzig (1954) for the shift scheduling problem first described in Edie (1954), can easily be adapted to the personalized multi-activity shift scheduling problem to obtain model D:

$$f(D) = \min \sum_{e \in E} \sum_{s \in \Omega^e} c_s^e x_s^e$$

$$\sum_{e \in E} \sum_{s \in \Omega^e} \delta_{ijs}^e x_s^e \geq b_{ij}, \qquad \forall i \in I, j \in J, \tag{6.1}$$

$$\sum_{s \in \Omega^e} x_s^e = 1, \qquad \forall e \in E, \tag{6.2}$$

$$x_s^e \in \{0,1\} \qquad \forall e \in E, s \in \Omega^e, \tag{6.3}$$

where $\delta_{ijs}^e = 1$ if work-activity $j \in J$ is assigned to period $i \in I$ in shift $s \in \Omega^e$, and variable $x_s^e = 1$ if employee e is assigned to shift $s \in \Omega^e$. The number of variables in model D grows rapidly with the number of employees and the number of feasible shifts per employee, which makes impractical to solve the model by generating all feasible shifts *a priori*.

In this paper, we present a column generation approach for solving model D, where the pricing subproblem is modeled by using a context-free grammar that allows to extract the set of feasible shifts for each employee. The pricing subproblem is solved efficiently by a dynamic programming algorithm performed on a directed acyclic graph obtained from the context-free grammar. This column generation method is embedded within a branch-and-bound algorithm. The resulting branch-and-price (B&P) algorithm can solve large-scale problem instances to near optimality. We show that

this algorithm is competitive with other methods from the literature, while allowing enough flexibility to address a variety of multi-activity shift scheduling problems.

The paper is organized as follows. In Section 6.2, we present a literature review on shift scheduling problems and we introduce grammar theory. In Section 6.3, we present our solution approach, including the grammar-based pricing subproblem and the branching rule used in the B&P algorithm. In Section 6.4, we present comparative computational results on two types of multi-activity shift scheduling problems presented in the literature.

6.2 Background Material

In this section, we present a literature review on models and methods for shift scheduling problems, in particular multi-activity shift scheduling problems. We then provide a brief introduction to context-free grammars, describing only the concepts that are relevant to the present work.

6.2.1 Literature Review

The literature on personnel scheduling distinguishes two problems: *shift scheduling* and *tour scheduling*. In shift scheduling problems, one is interested in determining the assignment of employees to one or more working activities, interspersed with breaks and meals, typically over a one-day planning horizon. In tour scheduling problems, complete schedules over several days have to be determined. Loucks and Jacobs (1991) and Ritzman *et al.* (1976) model the tour scheduling problem with assignment variables specifying the number of employees assigned to a given activity at any given time. Since such modeling approaches yield very large integer programming (IP) formulations, both papers propose heuristic methods. These models do not allow to place breaks or meals during the shifts, nor do they handle regulations concerning the transition between activities.

For shift scheduling problems, two types of IP models, explicit and implicit, are considered in the literature. In an *explicit* model, one obtains the schedule for each employee simply by scanning the optimal solution, i.e., in a time linear to the model size; the classical set covering formulation D is an example of an explicit model. In an *implicit* model, a post-processing algorithm, typically efficient, but not linear in the

model size, must be called upon in order to derive the schedule for each employee. The literature also distinguishes whether the problem has only one work-activity or multiple work-activities. In a *single-activity* shift scheduling problem, one only specifies, at each period, if an employee is working or not. In a *multi-activity* shift scheduling problem, there are several work-activities and whenever an employee is working at a given period, it is further necessary to specify which work-activity is assigned to that employee.

Problems involving a single work-activity and identical employees have been studied and efficiently solved to optimality using implicit models such as the ones suggested in Bechtold and Jacobs (1990); Aykin (1996) and Rekik *et al.* (2004). Implicit models have much less variables than the explicit set covering model (i.e., model D with $|J| = 1$), since the latter counts one variable per feasible shift, while the former use variables for types of shifts and types of breaks. Implicit models are limited in terms of the rules over shifts they can handle; in particular, we are aware of only one implicit model, the grammar-based model proposed in Côté *et al.* (2010), that can represent multi-activity shift scheduling problems. By contrast, the explicit set covering model (i.e., model D with an arbitrary set J) includes that case as well.

As mentioned above, the set covering model also has its limitations, since when the number of employees and the number of feasible shifts grow, the model can rapidly become intractable, unless a column generation approach, first introduced in Dantzig and Wolfe (1960), is used. Column generation is based on the idea that optimal solutions to large linear programs can be obtained without explicitly including all the columns (variables). The relevant columns can be generated dynamically by solving a so-called pricing subproblem (see for instance Desaulniers *et al.* (2005) and Lübbecke and Desrosiers (2005) and references therein for more details about column generation).

Up until now, very few papers addressed personalized multi-activity shift scheduling problems. Demassey *et al.* (2006) study a multi-activity shift scheduling problem on a 24-hour planning horizon involving up to ten work-activities. The set covering model is handled with a column generation approach in which the pricing subproblem is solved with constraint programming. This method succeeds in finding optimal integer solutions for some instances involving up to three work-activities, but not for larger instances. Lequy *et al.* (2009) consider a personalized muti-activity shift scheduling problem where shifts and breaks are fixed *a priori*. This paper describes

two IP models and a column generation approach based on multicommodity flow formulations. These exact approaches can deal with the smaller instances presented in the paper. To solve large-scale instances, a rolling horizon heuristic based on column generation (first studied in Omari (2002), Vatri (2001) and Bouchard (2004)) is used. In our computational experiments reported in Section 6.4, we will use the problem definitions and the instances from Demassey *et al.* (2006) and Lequy *et al.* (2009).

Formal languages have been used to derive IP models for multi-activity shift scheduling problems. Côté *et al.* (2009) propose two explicit IP models based on formal languages. Both models make use of assignment variables y_{ij}^{e} indicating whether or not employee e is assigned to activity $j \in J$ at period $i \in I$. One IP model is based on a regular language represented by a finite deterministic automaton that allows to encapsulate the constraints defining each feasible shift using a network flow formulation. Another IP model makes use of a context-free grammar that describes all feasible shifts for each employee. These models can handle personalized shift scheduling instances, but since they generate a large number of variables, they can only solve instances with few work-activities. In addition, in the case where many employees are alike, these models have symmetry issues. To deal with these performance issues, an implicit IP model based on context-free grammars was suggested in Côté *et al.* (2010) for multi-activity shift scheduling problems where all employees are identical. This model makes use of assignment variables y_{ij} indicating the number of employees assigned to activity $j \in J$ at period $i \in I$. To the best of our knowledge, this model is the first to solve efficiently large-scale multi-activity problem instances with up to ten work-activities, but it cannot be extended to personalized multi-activity problem instances. The present work is an attempt to fill this gap. Although we also use context-free grammars as in Côté *et al.* (2009) and Côté *et al.* (2010), our work differs significantly since we use the set covering model instead of models based on assignment variables. Moreover, we solve the pricing subproblem with a specialized dynamic programming algorithm; an alternative, tested in preliminary experiments and shown to be inferior, would consist in solving the pricing subproblem as an IP model similar to the one presented in Côté *et al.* (2009). Finally, instead of using a state-of-the-art IP software package to derive integer solutions, as in Côté *et al.* (2009) and Côté *et al.* (2010), we develop our own B&P implementation.

6.2.2 Definitions

Grammars, words and languages. A *context-free grammar* G is characterized by a tuple (Σ, N, P, S) where:

- Σ is an alphabet containing letters $(a, b, c, \dots)$, also called terminal symbols;
- N is a set of non-terminal symbols $(A, B, C, \dots)$;
- P is a set of productions of the form $X \to \alpha$, where $X \in N$ and α is a sequence of terminal and/or non-terminal symbols.
- S is the starting non-terminal.

A sequence of letters from alphabet Σ, called a *word*, is *recognized* by grammar G if it can be generated by successive applications of productions from G, starting with non-terminal S. The set of words recognized by a grammar is called a *language*.

In the following, we will use the term grammar to refer to a context-free grammar and we will assume that, except when specified otherwise, all grammars are in Chomsky normal form, meaning that all productions are of the form $X \to \alpha$ where $X \in N$ and $\alpha \in (N \times N) \cup \Sigma$. Note that this assumption is not restrictive since any context-free grammar can be converted to Chomsky normal form; see Hopcroft *et al.* (2001) for more information on formal languages.

Example 5 *The following grammar G defines all feasible shifts for a simple multi-activity shift scheduling problem. A shift must have a duration equal to the planning horizon and contain one break of one period anywhere during the shift except at the first or the last period. The problem contains two work-activities represented by letters j_1 and j_2 and break periods are represented by letter b. A break is mandatory to change from one work-activity to another.*
$G = (\Sigma = (j_1, j_2, b), N = (S, X, W, B, J_1, J_2), P, S)$, *where P is:*
$S \to WX, \qquad X \to BW,$
$W \to J_1J_1 \mid J_2J_2 \mid j_1 \mid j_2,$
$J_1 \to J_1J_1 \mid j_1,$
$J_2 \to J_2J_2 \mid j_2,$
$B \to b,$
*where the symbol $\mid$ specifies a choice of production. Assuming that the planning horizon has a duration of 4 periods, then the shifts $j_1bj_1j_1$, $j_2j_2bj_1$ and $j_1bj_2j_2$, among others, are recognized by G, while $j_1bj_1j_2$ is not. Word $j_1bj_2j_2$ is obtained by the derivation shown in Table 6.1, where **P** is the production used and **CS** is the current*

sequence, obtained from the previous sequence by applying the production on the left side.

Parse Trees. Any given word recognized by a grammar can be represented by a so-called *parse tree* for which the root node is associated with the starting non-terminal S and the leaves correspond to letters from Σ that form a word when listed from left to right. In a parse tree, the productions are represented as follows: an interior node represents a non-terminal on the left hand side of a production and its children are the non-terminals or the letter on the right hand side of the production. Figure 6.1 shows the parse tree associated with word $j_1bj_2j_2$ derived from grammar G of Example 5.

Grammar DAG. Quimper and Walsh (2007) suggest a way to generate a directed acyclic graph (DAG) embedding every parse tree corresponding to a word of length n recognized by a given grammar. The resulting DAG has two types of nodes: the or-nodes and the and-nodes. An or-node, associated with a non-terminal X, a position i and a length l, is the root of all the parse trees derived from X and giving a subsequence of length l starting at position i. An and-node represents a production starting with the non-terminal associated with its parent and resulting in a subsequence of length l starting at position i. Thus, any path in this DAG from the root-node to the leaves alternates between or-nodes and and-nodes. To derive any parse tree from the DAG, we start at the root-node. We visit an or-node by selecting exactly one child, which is necessarily an and-node. We visit an and-node by choosing all its children (exactly two if $l > 1$, one otherwise). By traversing the DAG in this way until the only remaining unvisited nodes are leaves, we obtain a parse tree associated with the word defined by the leaves. Conversely, starting from a given word ω, we can traverse the DAG backwards in a straightforward way to derive the parse tree associated with ω. In practice, the DAG is built by a procedure suggested in Quimper and Walsh (2007) inspired by an algorithm from Cooke, Younger, and Kasami (see Hopcroft *et al.* (2001)).

Figure 6.2 is the DAG derived from grammar G from Example 5 on words of length 4. The or-nodes are labeled O_{il}^{π} where π is a non-terminal or a letter associated with the node, while i and l are, respectively, the starting position and the length of the subsequence it produces. Usually, an and-node is labeled $A_{il}^{\Pi,t}$, where Π is a production $X \to \alpha$, i and l are, respectively, the starting position and the length

TABLE 6.1 Derivation of word $j_1bj_2j_2$ from grammar G of Example 5

P	**CS**
$-$	S
$S \rightarrow WX$	WX
$X \rightarrow BW$	WBW
$W \rightarrow J_2J_2$	WBJ_2J_2
$W \rightarrow j_1$	$j_1BJ_2J_2$
$B \rightarrow b$	$j_1bJ_2J_2$
$J_2 \rightarrow j_2$	$j_1bj_2J_2$
$J_2 \rightarrow j_2$	$j_1bj_2j_2$

of the subsequence generated from this production, and there is an index t for each such possible subsequence. To avoid overloading the figure, we did not label the and-nodes, which are simply illustrated with black dots, since it is easy to deduce the productions they represent from their parent and children nodes. For instance, the and-node having O_{12}^{W} as parent and $O_{11}^{J_1}$ and $O_{21}^{J_1}$ as children would be labeled $A_{12}^{W \rightarrow J_1 J_1, 1}$.

6.3 Grammar-Based Column Generation Approach

In this section, we present the main ingredients of our grammar-based B&P algorithm, namely the restricted master problem and the pricing subproblem solved at each iteration of the column generation approach, as well as the branching rule used to produce integer solutions.

6.3.1 Restricted Master Problem

At every node of the B&P tree, we solve the linear programming (LP) relaxations of a sequence of restrictions of model D, called the *restricted master problems*. Conceptually, each restriction is defined by allowing only a subset of the feasible shifts $\widetilde{\Omega}^e \subseteq \Omega^e$ for each employee $e \in E$. The sequence of restrictions is obtained by gradually enlarging the subsets of feasible shifts, which yields a sequence of non-decreasing lower bounds that converges towards the optimal LP relaxation of the node. More

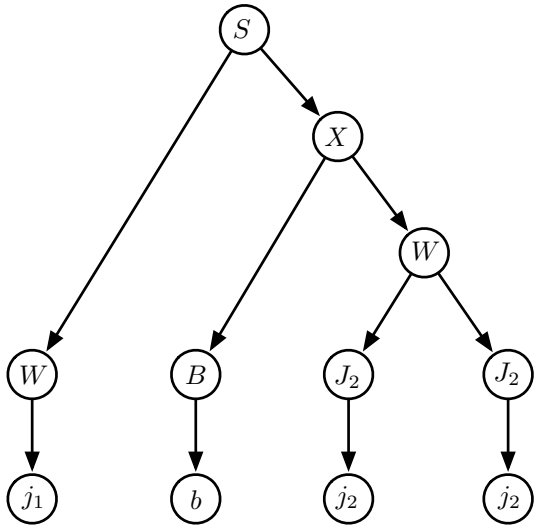

FIGURE 6.1 Parse tree for word $j_1 b j_2 j_2$ derived from grammar G of Example 5

precisely, at each iteration of the column generation method, we solve the current restricted master problem and then look for negative reduced cost columns, i.e., shifts in $\Omega^e \setminus \widetilde{\Omega}^e$ such that $\bar{c}_s^e < 0$ for each employee $e \in E$; these columns can be obtained by solving the pricing subproblem, which is described in the next subsection. If no negative reduced cost columns can be generated, the current restricted master problem is optimal and we have computed the LP relaxation of the node by generating only a (typically small) subset of the feasible shifts.

The restricted master problem, called RMP, solved at every iteration of the column generation method performed at each node of the B&P tree, takes the following form:

$$f(RMP) = \min \sum_{e \in E} \sum_{s \in \widetilde{\Omega}^e} c_s^e x_s^e$$

$$\sum_{e \in E} \sum_{s \in \widetilde{\Omega}^e} \delta_{ijs}^e x_s^e \geq b_{ij}, \qquad \forall i \in I, j \in J, \tag{6.4}$$

$$\sum_{s \in \widetilde{\Omega}^e} x_s^e = 1, \qquad \forall e \in E, \tag{6.5}$$

$$x_s^e \geq 0 \qquad \forall e \in E, s \in \widetilde{\Omega}^e. \tag{6.6}$$

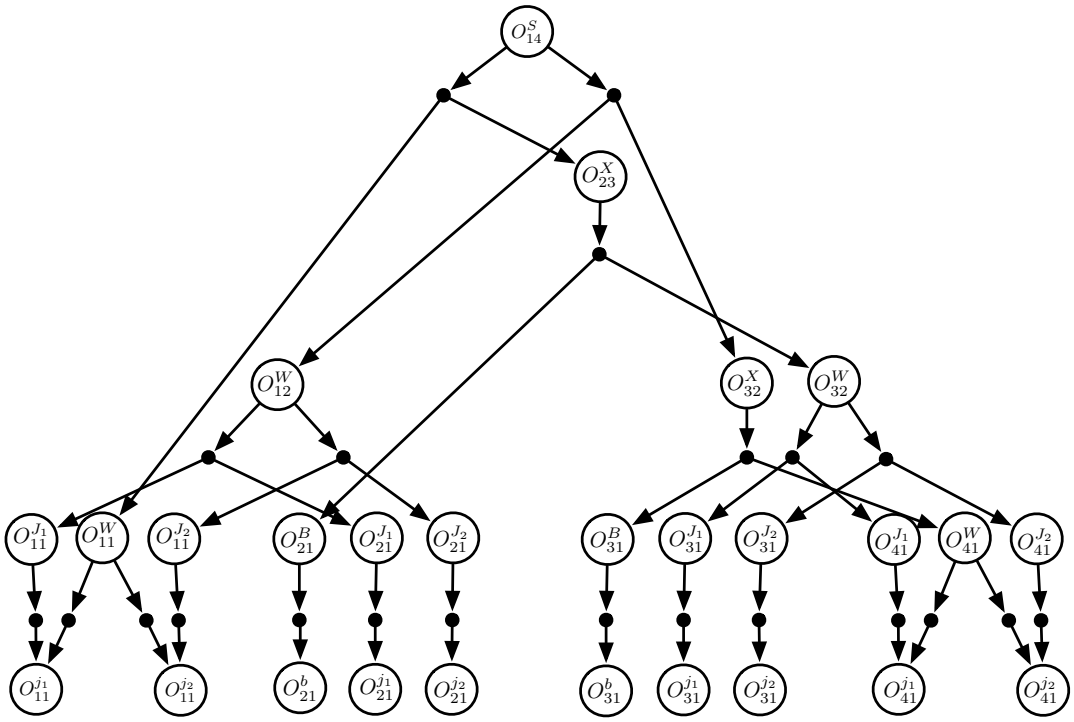

FIGURE 6.2 DAG for grammar G from Example 5 on words of length 4

To define each node of the B&P tree, we forbid some shifts to be assigned to a particular employee, as described in Subsection 6.3.3. Conceptually, this simply amounts to define the restricted subsets associated with that employee in such a way as to remove the corresponding forbidden shifts (in practice, however, we will associate a very large cost with these forbidden shifts, as seen in Subsection 6.3.3). Thus, the restricted master problems at every node of the B&P tree always take the above form and no additional constraints are needed to capture the branching rules that define the node.

6.3.2 Pricing Subproblem

To generate new columns to be included to the current RMP, we solve one subproblem for each employee, based on the DAG described in Section 6.2.2. For each employee $e \in E$, we generate a grammar G_e that represents the shifts employee e can

perform, according to the employee's skills, preferences and availability, and considering also general work regulations that apply to all employees. From this grammar, we generate the associated DAG that will be used to solve the pricing subproblem for employee e.

For the current RMP, let $\lambda_{ij} \geq 0$ be the dual variable associated with each constraint of type (6.4) and σ^e represent the (unrestricted) dual variable associated with each constraint of type (6.5). For the sake of clarity, we assume that the cost per shift can be decomposed by period and by work-activity as follows: $c_s^e = \sum_{i \in I} \sum_{j \in J_i^e} \delta_{ijs}^e c_{ij}^e$, where J_i^e is the set of work-activities employee e can perform at period i and c_{ij}^e is the cost for employee e to perform work-activity j at period i. At the end of this subsection, we discuss how we can generalize our approach to other, more realistic, cost structures. The reduced cost of column $s \in \Omega^e$ is then:

$$\bar{c}_s^e = \sum_{i \in I} \sum_{j \in J_i^e} (c_{ij}^e - \lambda_{ij}) \delta_{ijs}^e - \sigma^e \qquad \forall e \in E, s \in \Omega^e. \tag{6.7}$$

To solve the pricing subproblem for employee e, we associate a cost with each node of the DAG. To each leaf corresponding to a work-activity j at period i, we initialize its cost k_{ij} to $c_{ij}^e - \lambda_{ij}$. The other nodes of the graph have their cost initialized to zero. We solve each subproblem by a dynamic programming algorithm suggested in Quimper and Rousseau (2009) to find a minimum cost parse tree in a grammar-based DAG. The algorithm traverses the DAG from the leaves to the root by summing up the children of the and-nodes and by choosing the lowest cost children of an or-node (see the updating formulae below). As a result, every child of the root node with a negative value represents a negative reduced cost column that can be added to RMP. If no such child exists in any of the employee subproblems, no negative reduced cost column can be generated and the current RMP solution is the optimal LP solution.

To present the dynamic programming updating formulae, we use the notations introduced in Section 6.2.2 to refer to the or-nodes and the and-nodes of the DAG. Let $cost_O(N)$ and $cost_A(N)$ be, respectively, the costs associated with or-node $O(N)$ and and-node $A(N)$; also, let $ch(N)$ be the children of or-node N. We then update

the costs according to the following formulae:

$$cost_O(O^j_{i1}) = \begin{cases} k_{ij}, & \text{if } j \in J^e_i, \\ 0, & \text{otherwise,} \end{cases} \tag{6.8}$$

$$cost_O(O^\pi_{il}) = \min_{A^{\Pi,t}_{il} \in ch(O^\pi_{il})} \left\{ cost_A(A^{\Pi,t}_{il}) \right\} \qquad l > 1 \tag{6.9}$$

$$cost_A(A^{\Pi:B \to j,t}_{i1}) = cost_O(O^j_{i1}) \tag{6.10}$$

$$cost_A(A^{\Pi:B \to CD,t}_{il}) = cost_O(O^C_{ik}) + cost_O(O^D_{i+k,l-k}) \qquad l > 1 \tag{6.11}$$

where B, C, D are non-terminal symbols of the grammar, C corresponding to a subsequence of length $k < l$.

Note that the assumption restricting the cost to be $c^e_s = \sum_{i \in I} \sum_{j \in J^e_i} \delta^e_{ijs} c^e_{ij}$ is not necessary for our algorithm to work, as we could have cost on any node of the DAG associated with the grammar. For instance, a cost could be assigned to every and-node associated with a production, representing a transition cost between different work-activities. In the dynamic programming algorithm, these and-nodes would be initialized to this transition cost and this cost would be added to the total cost of the node, when processed.

6.3.3 Branching Rule

Since the optimal solution to the final RMP at any node of the B&P tree is likely to be fractional, we need to perform branching in order to find an optimal IP solution to model D. Branching on individual x^e_s variables by generating the two nodes $x^e_s = 1$ and $x^e_s = 0$ is not a good idea: while the former node is easily dealt with, the latter cannot be easily handled in the pricing subproblem. Therefore, we must develop another type of branching rule that not only eliminates the current fractional solution, but that can also be easily processed by the dynamic programming algorithm used to solve the pricing subproblem.

We suggest the following rule, adapted from the B&P algorithm of Barnhart *et al.* (2000) for solving integer multicommodity flow problems. First, we select an employee e' such that there exists at least two associated variables having fractional values in the optimal LP solution. For employee e', we select the two shifts $s^{e'}(1)$ and $s^{e'}(2)$ corresponding to the associated variables with the highest fractional values. We then identify the first divergent position, meaning the first period at which shifts $s^{e'}(1)$

and $s^{e'}(2)$ differ in terms of their work-activities. If we denote by i' the first divergent position and by $j(1)$ and $j(2)$, respectively, the two work-activities assigned to $s^{e'}(1)$ and $s^{e'}(2)$ at period i', we then generate a partition of $J_{i'}^{e'}$ into two subsets $J_{i'}^{e'}(1)$ and $J_{i'}^{e'}(2)$ such that $j(l) \in J_{i'}^{e'}(l)$, for $l = 1, 2$. Apart from this rule involving $j(1)$ and $j(2)$, the other work-activities in $J_{i'}^{e'}$ are included arbitrarily in one of the two subsets, but in such a way that both subsets have the same number of elements (up to a difference of one). Finally, we generate two nodes in the B&P tree: each one forbids solutions where employee e' performs a work-activity in $J_{i'}^{e'}(l)$ at period i', for $l = 1, 2$. Note that $J_{i'}^{e'}$, the set of work-activities employee e' can perform at period i', can vary from one branch and bound node to another according to branching decisions taken higher up in the tree.

This rule ensures a well-balanced tree and can be easily handled in both the restricted master problems and the pricing subproblems. Indeed, it suffices to assign a very large value to the cost $c_{i'j}^{e'}$ associated with forbidden work-activities $j \in J_{i'}^{e'}(l)$, for $l = 1, 2$. This will effectively remove the corresponding shifts from RMP, thus eliminating the current fractional solution. When solving the pricing subproblem for employee e', this modification corresponds to assigning a very large cost to each leaf representing position i' and work-activities $j \in J_{i'}^{e'}(l)$, for $l = 1, 2$, which ensures that these leaves will not be selected by the dynamic programming algorithm.

In many problem instances, the beginning of the shifts, the shift lengths and the breaks are not fixed *a priori*. In this case, the activities at the divergent position are not necessarily work-activities, but can also be rest-activities (rest, breaks or meals). The branching scheme must then be adapted by simply adding the rest-activities to the set of work-activities, so that large cost values can be assigned to the corresponding forbidden rest-activities, in both the restricted master problem and the pricing subproblem.

6.4 Computational Experiments

In this section, we compare our grammar-based B&P algorithm with existing methods, using the problem definitions and the instances from Demassey *et al.* (2006) and Lequy *et al.* (2009). For the two classes of problems, we describe the problem and the associated grammars, and we provide computational results comparing our approach with available results in the literature. The experiments on our B&P code

were performed (in sequential) on a two-processor quad-core intel Xeon 2.4GHz with 48 GB RAM. To solve the restricted master problems, we use the barrier method in CPLEX 11.2, with all parameters kept at their default values. The branching rule and the column generation method at each node of the B&P tree, including the dynamic programming algorithm, are implemented in C++ and embedded in a B&P algorithm using the OOBB framework from Crainic *et al.* (2009).

6.4.1 Problem Instances From Demassey *et al* (2006)

This section presents a shift scheduling problem for a retail store, allowing up to ten different work-activities. For each number of work-activities (1 to 10), ten instances are available. They differ in their demand curves, number of employees and costs. All employees are assumed to be identical; although our algorithm is specifically designed for personalized shift scheduling problems, it is important to show that it can be easily adapted to the case of identical employees, and still remain competitive with existing methods even in that case. We present the specifications of the problem and then compare our approach to the column generation method from Demassey *et al.* (2006) and to the formal language based models from Côté *et al.* (2009) and Côté *et al.* (2010) tested on these problem instances.

Problem Definition

Given:

- a 24-hour planning horizon divided into 15-minute periods;
- for each work-activity and each period:
  - the required number of employees;
  - unit undercovering and overcovering costs;
  - a cost to perform the work-activity at the given period;
- the number of employees;

the problem is to assign one shift to each employee such that:

- a shift may start at any period of the day allowing enough time to complete its duration during the planning horizon;
- a shift must cover between 3 hours and 8 hours of work-activities;
- if a shift covers at least 6 hours of work-activities, it must have two 15-minute breaks and a lunch break of 1 hour;

- if a shift covers less than 6 hours of work-activities, it must have one 15-minute break, but no lunch;
- if performed, the duration of a work-activity is at least 1 hour (4 consecutive periods);
- a break (or a lunch) is necessary between two different work-activities;
- work-activities must be inserted between breaks, lunch and rest stretches;

while minimizing:

- the total cost of the assigned shifts (the cost of a shift is the sum over all periods of the costs of all work-activities performed in this shift) and
- the total overcovering and undercovering costs.

Adapting the Solution Approach. Although all employees are identical, we could solve the above problem directly as a personalized multi-activity shift scheduling problem with our B&P algorithm. The performance of the algorithm would be seriously impaired for instances with a large number of employees. We therefore adapt our solution approach to address this issue. At the root node, we define a different restricted master problem, RMP', that aggregates employees instead of considering each of them individually. More specifically, we define Ω to be the set of feasible shifts for any employee and $\delta_{ij}^s = 1$ if work-activity j is assigned to period i in shift $s \in \Omega$. Each variable x_s of the aggregated model represents the number of employees assigned to shift $s \in \Omega$. Finally, we introduce $\widetilde{\Omega} \subseteq \Omega$ the subset of allowed feasible shifts in RMP', which is defined as follows:

$$f(RMP') = \min \sum_{s\in\widetilde{\Omega}} (\sum_{i\in I} \sum_{j\in J} \delta_{ijs} c_{ij}) x_s + \sum_{i\in I} \sum_{j\in J} (c_{ij}^+ s_{ij}^+ + c_{ij}^- s_{ij}^-)$$

$$\sum_{s\in\widetilde{\Omega}} \delta_{ijs} x_s - s_{ij}^+ + s_{ij}^- = b_{ij}, \qquad \forall i \in I, j \in J, \tag{6.12}$$

$$\sum_{s\in\widetilde{\Omega}} x_s = \mid E \mid, \tag{6.13}$$

$$x_s \geq 0, \qquad \forall s \in \widetilde{\Omega}, \tag{6.14}$$

$$s_{ij}^+, s_{ij}^- \geq 0, \qquad \forall i \in I, j \in J, \tag{6.15}$$

where $\mid E \mid$ is the number of employees and c_{ij}, c_{ij}^+ and c_{ij}^- are, respectively, the cost,

the unit overcovering cost and the unit undercovering cost associated with work-activity j at period i. Variables s_{ij}^{+} and s_{ij}^{-} correspond, respectively, to the number of employees overcovering or undercovering the demand for work-activity j at period i. A similar model is used in the column generation approach of Demassey *et al.* (2006).

At the root node of the B&P tree, every iteration of the column generation method first consists in solving RMP' and then in searching for negative reduced cost columns by a single application of the dynamic programming algorithm of Section 6.3.2. At other nodes of the B&P tree, every iteration of the column generation approach proceeds as described in Section 6.3, using the restricted master problems of the form RMP, which are initialized by simply copying for each employee the set of columns generated at the root node. Note that this approach differs significantly from the column generation algorithm in Demassey *et al.* (2006) which uses aggregated restricted master problems throughout the whole solution process.

Definition of the Grammar. The following presents the grammar used for this problem. For the sake of clarity, the grammar is not stated in Chomsky normal form.
$G = (\Sigma = (a_j \ \forall j \in J, r), N = (S, F, P, W, A_j \ \forall j \in J, B, L, R), P, S)$,
where a_j is a period assigned to work-activity $j \in J$ and r is a period assigned to any rest-activity (rest, lunch or break). In P, defined as follows, the notation $\rightarrow_{[min,max]}$ is used to restrict the subsequences generated with a given production to have a length between min and max periods:

$$
\begin{array}{ll}
S \rightarrow RFR \mid FR \mid RF \mid RPR \mid PR \mid RP & B \rightarrow r \\
F \rightarrow_{[30,38]} WBWLWBW \mid WLWBWBW \mid WBWBWLW & L \rightarrow rrrr \\
P \rightarrow_{[13,24]} WBW & R \rightarrow Rr \mid r \\
W \rightarrow_{[4,\infty)} A_j \quad \forall j \in J & \\
A_j \rightarrow A_j a_j \mid a_j \quad \forall j \in J &
\end{array}
$$

Computational Results. We first compare our approach on the instances described above with the results reported in Demassey *et al.* (2006). Experiments in Demassey *et al.* (2006) were run on an Opteron 250. Table 6.2 presents average statistics on the ten classes of instances (a class of instances contains the ten instances with the same number of work-activities). *NbA* is the number of work-activities in the class of instances. The *Root node* columns display *Time(s)*, *NbIts* and *NbCols*, respectively, the CPU time in seconds, the number of column generation iterations and the number

of columns generated to solve the LP relaxation at the root node. The *Branch-and-Price* columns show the following statistics: $NbS(0.01\%)$, the number of instances (out of ten) solved within a 0.01% relative gap (i.e., $Gap = 100(Z^u - Z^l)/Z^u$ where Z^u and Z^l are, respectively, the best upper and lower bounds) within a CPU time limit of 2 hours; *Time(s)*, the average CPU times for the instances solved within a 0.01% relative gap; $NbS(1\%)$, the number of instances solved within a 1% relative gap within a CPU time limit of 2 hours. Note that Demassey *et al.* (2006) do not report any gaps.

These results clearly show that our algorithm can solve the root node LP relaxation faster, in fewer iterations and with less columns than the method of Demassey *et al.* (2006). In addition, our branching rule also performs better on these instances, allowing our B&P algorithm to solve to near optimality instances with up to ten work-activities within 2 hours of CPU time. We observe that for the majority of the 46 instances that are not solved to the 0.01% gap tolerance, the gaps are quite small at the end of the 2h CPU time limit. Indeed, 36 instances have a gap between 0.01% and 1% at the end of the time limit.

The next table compares our B&P algorithm to other approaches based on formal languages. Table 6.3 displays the CPU times for the one- and two-activity instances (ten in each class, *No.* being the instance number) for the explicit formulations from Côté *et al.* (2009) based on automata (*IP R M*) and grammars (*IP G M*), for the implicit grammar-based model (*IG M*) from Côté *et al.* (2010), and for our B&P algorithm (*G-B CG*). The CPU times (in seconds) are the times to reach a 1% relative gap. The notation $> 1h$ means that the instances could not be solved to the gap tolerance within the 1-hour time limit. Experiments in Côté *et al.* (2009) were run on a 2.4 GHz Dual AMD Opteron Processor 250 with 3 GB of RAM using CPLEX 10.0. Experiments in Côté *et al.* (2010) were performed on a 2.3GHz AMD Opteron with 3GB of memory using CPLEX 10.1.1.

These results show that the grammar-based column generation method is competitive with the existing approaches based on formal languages, at least for problem instances with up to two work-activities. While it can solve to near optimality many instances involving up to ten work-activities, as shown in Table 6.2, our B&P algorithm is generally outperformed by the implicit grammar-based model from Côté *et al.* (2010) on instances from three to ten work-activities. This is only mildly surprising, since the implicit grammar-based model from Côté *et al.* (2010) totally avoids

TABLE 6.2 Comparison with Demassey *et al.* (2006) approach

	Root Node			Branch and Price		
NbA	$Time(s)$	$NbIts$	$NbCols$	$NbS(0.01\%)$	$Time(s)$	$NbS(1\%)$
Grammar-based CG						
1	0.1	10	257	5	62	10
2	0.2	16	601	6	100	9
3	0.6	20	975	6	2074	8
4	1.1	25	1321	5	2096	9
5	3.0	46	2321	0	$> 2h$	10
6	4.0	47	2457	9	915	10
7	6.6	47	3117	5	2426	9
8	8.3	62	3459	7	2163	10
9	9.6	62	3626	5	1886	7
10	9.9	43	3405	6	3754	8
Demassey *et al.* (2006) CG						
1	0.4	19	889	8	144	–
2	3.7	48	2340	8	394	–
3	2.0	52	2550	4	1592	–
4	12.5	103	5063	0	$> 2h$	–
5	6.2	86	4288	0	$> 2h$	–
6	13.8	130	6493	0	$> 2h$	–
7	18.4	137	6839	0	$> 2h$	–
8	25.4	155	7736	0	$> 2h$	–
9	25.9	155	7741	0	$> 2h$	–
10	42.0	179	8974	0	$> 2h$	–

TABLE 6.3 Comparison between approaches based on formal languages on instances with one and two work-activities-CPU time(s)

No.	G-B CG	IP R M	IP G M	IG M
One work-activity				
1	0.02	1.03	7.42	0.26
2	373.01	40.09	$> 1h$	110.88
3	3.32	64.64	$> 1h$	75.25
4	1.78	46.39	1850.38	2.75
5	0.13	14.03	322.57	0.48
6	0.03	3.28	130.21	0.34
7	1.53	5.99	1662.75	2.71
8	30.75	131.77	$> 1h$	2642.12
9	1.87	16.14	1015.10	1.18
10	0.84	20.22	1313.28	0.80
Two work-activities				
1	0.27	228.07	2826.4	1.27
2	3.51	2870.20	1952.58	4.12
3	13.63	1541.15	$> 1h$	81.91
4	25.66	169.96	$> 1h$	16.27
5	0.32	$> 1h$	$> 1h$	2.59
6	6.98	1288.56	$> 1h$	51.16
7	1.88	29.94	$> 1h$	0.60
8	23.30	$> 1h$	325.08	36.20
9	$> 1h$	$> 1h$	$> 1h$	$> 1h$
10	0.97	1108.23	$> 1h$	4.99

symmetry issues and does not suffer from the growth in the number of employees, contrary to our B&P algorithm. The implicit grammar-based model cannot, however, deal with *personalized* multi-activity shift scheduling problems. We now present computational results on such problems.

6.4.2 Problem Instances From Lequy *et al* (2009)

This section presents computational results on a personalized multi-activity shift scheduling problem introduced by Lequy *et al.* (2009) under the name *multi-activity assignment problem*. Two sets of instances are available for this problem: results on the first set are published in Lequy *et al.* (2009), while results on the second set are still unpublished, but were made available to us by the authors. These results are obtained from experiments performed on an $IntelCore^{TM}2\ CPU$ 6700 clocked at 2.66GHz with 4GB RAM using the Xpress-MP solver.

Problem Definition

Given:

- a planning horizon divided into 15-minute periods;
- for each work-activity;
  - the required number of employees at each period;
  - the undercovering and overcovering costs;
  - its minimum and maximum durations;
- for each available employee;
  - the list of pre-assigned work-pieces (a work-piece is defined by a starting time and a duration);
  - the list of work-activities for which the employee is qualified;

the problem is to fill each work-piece with a sequence of activities such that:

- each employee can only be assigned to work-activities for which he is qualified;
- the minimum and maximum work-activity durations are satisfied;

while minimizing:

- the total undercovering and overcovering costs and
- the total transition costs (a cost is associated with every transition from one work-activity to another within a work-piece).

Definition of the Grammar. The following presents the grammar used for this problem for a given employee e and a given work-piece p. For the sake of clarity, as before, the grammar is not stated in Chomsky normal form.
$G^{e,p} = (\Sigma = (a_j \ \forall j \in J_e), N = (S, \{A_j, A_j^n, A_j'\} \ \forall j \in J_e), P, S)$,
where J_e is the set of work-activities for employee e and a_j is a period assigned to work-activity $j \in J_e$. To define P, we use the following notations: $\rightarrow_{[min,max]}$ restricts the subsequences generated with a given production to have a length between min and max periods; l_{ep} is the length of work-piece p for employee e; min_j and max_j are, respectively, the minimum and maximum durations of work-activity j. P is then defined as follows:

$\forall j \in J_e$:

$$
\begin{array}{lll}
S \rightarrow_{[l_{ep},l_{ep}]} A_j A_j^n & & \\
S \rightarrow_{[l_{ep},l_{ep}]} A_j & \text{if } l_{ep} \leq max_j & \\
A_j \rightarrow_{[min_j,max_j]} A_j' & & \\
A_j^n \rightarrow A_{j'} A_{j'}^n \mid A_{j'} & \forall j' \in J_e \setminus \{j\} & \\
A_j' \rightarrow A_j' a_j \mid a_j & &
\end{array}
$$

Computational Results. On the first set of instances, we compare our grammar-based column generation (*G-B CG*) method with three approaches from Lequy *et al.* (2009): two models solved exactly by the IP solver Xpress-MP and the rolling horizon heuristic method based on column generation (*Horizon CG*). The first model is a multicommodity network flow model (*MC model*), while the second is a reformulation of the first that yields fewer variables (*Block model*). The heuristic method is based on a rolling horizon framework where each time slice is solved with a column generation approach based on a shortest path subproblem. Integer solutions are found with a rounding procedure that iteratively fixes variables to integer values, each time reoptimizing the resulting LP relaxation by column generation.

Table 6.4 presents the comparative times and solution values for our B&P algorithm and the two exact approaches from Lequy *et al.* (2009) on the smallest instances, where $No.$ is the instance number and $Value$ is the value of the solution found by the associated approach in $Time(s)$ seconds. The notation $> 1h$ means that the optimality could not be proved within the 1-hour CPU time limit. In these cases, $Value$ is the best integer solution found within this time limit. For the largest instances, Lequy *et al.* (2009) only reports times in seconds to find the first integer solution

with the *Block model*. Table 6.5 compares these times with the time necessary for the B&P algorithm to find the first integer solution. For our approach, we also present the value of the first integer solution (Val) found and the relative gap between this solution and the best solution found for this instance ($Gap(\%) = 100(Z_F - Z_B)/Z_F$, where Z_F is the first integer solution found and Z_B is the best solution reported in Table 6.6). The value of the first integer solution found with the *Block model* is not reported in Lequy *et al.* (2009). Table 6.6 presents the comparative results between our approach and the *Horizon CG* method on all the instances. To perform a fair comparison with the heuristic method, we stopped the B&P algorithm when a solution within a 1% relative gap was found. $> 1h$ means that this gap could not be achieved within the 1-hour CPU time limit. In these cases, $Value$ is the best integer solution found within the time limit. In bold we highlight the times and values for the instances where our approach is strictly better than *Horizon CG*.

The results presented in Table 6.4 show that our approach is generally competitive with the *MC model* and the *Block model* on smaller instances, except for a few instances, where it is clearly outperformed. For larger instances, however, our method finds a first integer solution much faster than the *Block model*, as shown in Table 6.5. The first integer solution found by our method is also of extremely good quality. In Table 6.6, when compared to the heuristic method *Horizon CG*, our method is not competitive in terms of CPU times, but for 16 out of 20 instances, it finds a better solution, still in reasonable time.

Table 6.7 compares our approach with the *Horizon CG* heuristic method on the second set of instances. As in Table 6.6, $No.$ is the instance number and $Value$ is the value of the solution found by the associated approach in $Time(s)$ seconds. For the *G-B CG* approach, we stopped when a solution within a 1% relative gap was found. No time limit was applied for these instances. As before, we highlight in bold the times and values for the instances where our approach is strictly better than *Horizon CG*.

On the second class of instances, as the results in Table 6.7 show, our B&P algorithm generally outperforms *Horizon CG*, not only in solution quality, but also in CPU times. Indeed, for all instances, the B&P algorithm finds solution values equal or better than those obtained by method *Horizon CG* and it does so in less time for 17 out of the 20 instances.

TABLE 6.4 Comparison with exact methods on the smallest instances of the first set of instances of Lequy *et al.* (2009) problem

	G-B CG		MC model		Block model	
No.	$Value$	$Time(s)$	$Value$	$Time(s)$	$Value$	$Time(s)$
7 days, 20 employees, and 5 activities						
1024	7220	14	7220	114	7220	17
1773	6345	9	6345	50	6345	17
2732	7420	94	7420	200	7420	21
4657	6400	2591	6400	129	6400	135
5553	7535	21	7535	92	7535	34
1 day, 50 employees, and 10 activities						
1808	3270	$> 1h$	3250	2681	3250	1190
5066	2440	947	2440	716	2440	294
5135	2580	103	2580	155	2580	98
5226	2725	34	2725	107	2725	93
8854	2800	$> 1h$	2740	839	2740	321

TABLE 6.5 Comparison with the *Block model* to find the first integer solution on the largest instances of the first set of instances of Lequy *et al.* (2009) problem

No.	G-B CG			Block model
	$Time(s)$	Val	$Gap(\%)$	$Time(s)$
7 days, 50 employees, and 10 activities				
1007	1313	14115	0.00	16108
156	1787	13420	0.00	21271
237	1439	13610	0.15	16050
4369	1530	13675	0.00	25029
5216	1811	14800	0.00	13020
2 days, 75 employees, and 12 activities				
1855	1504	6325	2.21	18685
2106	947	6525	0.00	15400
2435	734	6050	0.00	3511
4225	864	6270	0.24	18584
9863	624	5870	0.00	19216

TABLE 6.6 Comparison with the *Horizon CG* approach on the first set of instances of Lequy *et al.* (2009) problem

	G-B CG		Horizon CG	
No.	$Value$	$Time(s)$	$Value$	$Time(s)$
7 days, 20 employees, and 5 activities				
1024	**7220**	11	7265	10
1773	**6360**	9	6440	9
2732	7420	19	7420	17
4657	**6400**	2003	6415	11
5553	7600	15	7550	15
1 day, 50 employees, and 10 activities				
1808	**3270**	$> 1h$	3315	86
5066	**2440**	935	2565	237
5135	**2580**	**8**	2595	18
5226	**2725**	**8**	2740	19
8854	2800	$> 1h$	2770	70
7 days, 50 employees, and 10 activities				
1007	**14115**	1321	14265	363
156	**13420**	1793	13570	612
237	13610	$> 1h$	13590	333
4369	**13675**	1536	13890	473
5216	**14800**	1824	15040	543
7 days, 100 employees, and 15 activities				
1855	6265	$> 1h$	6185	1068
2106	**6525**	$> 1h$	6630	878
2435	**6050**	$> 1h$	6090	252
4225	**6255**	$> 1h$	6410	655
9863	**5870**	$> 1h$	6035	388

TABLE 6.7 Comparison with the *Horizon CG* approach on the second set of instances of Lequy *et al.* (2009) problem

	G-B CG		Horizon CG	
No.	$Value$	$Time(s)$	$Value$	$Time(s)$
7 days, 20 employees, and 5 activities				
1024	2940	**0.80**	2940	1.99
1773	2770	**0.76**	2770	2.17
2732	3820	**0.54**	3820	1.37
4657	3210	**0.61**	3210	2.21
5553	3270	**0.59**	3270	1.51
1 day, 50 employees, and 10 activities				
342	**1875**	**26.70**	1950	55.52
369	**2315**	146.08	2360	23.59
71	2050	**1.26**	2050	5.37
737	**2065**	53.23	2105	16.43
869	**1875**	**41.67**	1890	107.33
7 days, 50 employees, and 7 activities				
5600	**8440**	**59.49**	8500	98.05
592	**7345**	**33.03**	7375	130.09
8597	**7645**	**31.58**	7705	126.58
9445	7900	**14.67**	7900	42.42
949	**8155**	**44.19**	8185	132.59
7 days, 100 employees, and 15 activities				
530	**15200**	**5818.36**	15275	6667.41
1024	**15420**	**4602.99**	15690	6889.64
2596	15855	10806.10	15855	5805.99
6384	**15250**	**2064.81**	15400	4461.14
7862	**15940**	**1391.95**	16030	3545.54

When comparing the last two tables, we observe that our results on the two sets of instances are very contrasting. Indeed, we observe two main differences between the two sets of instances. First, each employee has more skills in the first set than in the second set, so each of them is allowed to perform almost all work-activities. Second, the work-pieces are, on average, of larger size in the first set than in the second set. These two characteristics yield a greater number of feasible shifts for each employee in the first set than in the second set. Our B&P is clearly sensitive to the number of feasible shifts per employee. Also, since each employee can perform almost all work-activities in the first set of instances, the employees are almost identical, which yields symmetry issues.

6.4.3 Summary

The previous sections presented computational results comparing our B&P algorithm with different approaches from Demassey *et al.* (2006), Côté *et al.* (2009), Lequy *et al.* (2009) and Côté *et al.* (2010).

From a modeling point of view, the grammar-based column generation approach is generic, as it can handle a variety of multi-activity shift scheduling problems, in particular personalized instances. By contrast, the column generation method of Demassey *et al.* (2006) and the implicit grammar-based model of Côté *et al.* (2010) can easily model multi-activity instances, but both approaches cannot be extended to the personalized case. We have also shown that the use of grammars can easily adapt to the context introduced in Lequy *et al.* (2009), where the breaks, the shift beginnings and the shift lengths are known *a priori*. The models and methods in Lequy *et al.* (2009) were developped precisely for this problem and cannot be adapted easily to another context.

From a computational point of view, the grammar-based column generation approach is flexible enough to solve efficiently a variety of problem instances. Indeed, the B&P algorithm provides integer solutions of good quality in reasonable computational time for almost all tested instances. In particular, on personalized problem instances from Lequy *et al.* (2009), it is competitive with a specialized heuristic method, showing superior performance on some classes of instances.

6.5 Conclusion

In this paper, we presented a B&P algorithm to solve different types of personalized multi-activity shift scheduling problems. The algorithm solves the LP relaxation of the classical set covering formulation with column generation. The pricing subproblem is modeled with a context-free grammar and solved with a dynamic programming algorithm based on traversing the DAG associated with the grammar. The B&P algorithm also integrates a branching rule that preserves the structure of the pricing subproblem. Our computational experiments show that the B&P algorithm is competitive with existing approaches from the literature. Furthermore, it is flexible enough to address different types of problems. This characteristic is mostly due to the expressiveness of grammars that enables to encode a large set of rules over shifts.

Acknowledgments

This work was supported by a grant from the Fonds québécois de recherche sur la nature et les technologies. We would like to thank the following individuals: François Guertin for allowing us to use the OOBB framework and for his help during the developpement of the algorithm; Serge Bisaillon for his support on many technical aspects throughout the progress of this work; Quentin Lequy for providing us with his two sets of instances and his related results.

Chapitre 7

DISCUSSION GÉNÉRALE

Dans les articles présentés aux chapitres 4, 5 et 6, nous avons d'abord introduit deux approches génériques de modélisation en nombres entiers basées sur des variables d'affectation binaires permettant d'aborder les problèmes de planification de quarts multi-activités. Ensuite, nous avons adapté une de ces idées de modélisation au contexte où les employés sont identiques. Finalement, nous avons proposé une méthode exacte, basée sur les mêmes fondements, appropriée pour les problèmes de planification de quarts multi-activités personnalisés. Par ces trois contributions, nous couvrons un large spectre de problèmes de planification de quarts de travail qui étaient restés pratiquement absents de la littérature.

De plus, l'utilisation des langages formels pour la formulation de restrictions sur des quarts est une idée qui avait certes été utilisée en programmation par contraintes, mais dans le domaine de la programmation mathématique, il s'agit d'un nouvel outil. C'est d'ailleurs un outil qui facilite la modélisation par son expressivité relativement proche d'un langage naturel, mais qui, surtout, permet de modéliser des problèmes d'une manière générique. L'addition d'une contrainte à un problème peut possiblement se faire en modifiant la grammaire avec l'ajout ou la modification de quelques productions et les modèles sous-jacents sont regénérés automatiquement.

Les comparaisons théoriques et expérimentales de nos modèles et méthodes de résolution avec les méthodes présentes dans la littérature montrent qu'en plus de permettre la modélisation de problèmes variés de planification de quarts, ils sont pertinents en pratique.

Tout au long de nos travaux de recherche nous avons testé nos approches sur un problème de planification de quarts multi-activités introduit par Demassey *et al.* (2006). Ce problème présente plusieurs formes de flexibilité dans la composition des quarts, notamment, des débuts et des fins de quarts variables, une affectation de

pauses dépendant du nombre d'heures travaillées, des limitations sur les séquences de travail consécutif et jusqu'à 10 activités de travail. Puisqu'il s'agit d'un problème de planification de quarts anonymes, nous avons pu utiliser autant nos modèles basés sur des variables d'affectations binaires, notre modèle implicite que notre approche de génération de colonnes pour l'aborder. Bien qu'en conclusion le modèle implicite soit nettement dominant sur ce problème, permettant d'éliminer toute forme de symétrie venant des employés identiques, ces expérimentations ont permis de faire ressortir différentes caractéristiques de chaque approche.

Premièrement, quoique peu performants lorsque le nombre d'activités augmente, les modèles du chapitre 4 se révèlent être des bonnes bases pour la modélisation. En pratique, le modèle basé sur les grammaires est moins prometteur que celui basé sur les automates, mais son intérêt vient du fait qu'il permet d'inclure toutes les règles du problème dans une seule grammaire et évite ainsi des contraintes supplémentaires. C'est cette caractéristique qui permet d'utiliser les grammaires à la base du modèle implicite du chapitre 5 et de l'algorithme de programmation dynamique sur lequel repose la méthode de génération de colonnes du chapitre 6.

Deuxièmement, malgré les limitations des formulations des chapitres 4 et 6 en ce qui concerne les symétries et la croissance du nombre d'employés, ces modèles pourraient aborder sans modification profonde une version personnalisée du problème de Demassey *et al.* (2006), d'où leur pertinence par rapport au modèle implicite basé sur les grammaires.

Finalement, les modèles du chapitre 4 basés sur des variables d'affectation binaires semblent à première vue, ne présenter qu'un intérêt théorique étant sans surprise dominés par la formulation de type recouvrement et la méthode de génération de colonnes. Par contre, il faut observer que d'un point de vue modélisation, ils permettent d'aborder un plus grand nombre de problèmes que la méthode de génération de colonnes du chapitre 6 pour lequel l'algorithme sous-jacent est restreint à ce que toutes les contraintes sur la formation des quarts de travail soient incluses dans la grammaire, ce qui n'est pas le cas du modèle en variables binaires basées sur les grammaires.

Pour mettre nos modèles et méthodes en parallèle avec ceux qui existent dans la littérature, nous avons aussi effectué des expérimentations sur d'autres problèmes. En premier lieu, nous avons pu, bien sûr, se comparer à la méthode de Demassey

et al. (2006) sur le problème mentionné ci-haut. Aussi, nous avons comparé notre modèle implicite avec un modèle de recouvrement classique et un modèle implicite sur une banque de problèmes anonymes et mono-activité regroupée par Mehrotra *et al.* (2000). Finalement, nous avons modélisé un problème multi-activités personnalisé introduit par Lequy *et al.* (2009) pour comparer notre approche de génération de colonnes avec deux modèles d'affectation et une méthode heuristique. Chacune de ces expérimentations montre la versatilité de nos approches à aborder plusieurs types de problèmes de planification de quarts et les résultats démontrent l'intérêt de celles-ci en comparaison avec des méthodes spécifiques.

Chapitre 8

CONCLUSION

Dans la présente thèse, nous avons abordé différents aspects du problème de planification de quarts de travail à l'aide de quelques modèles et méthodes basés sur les langages formels. Dans ce qui suit, nous ferons une synthèse de ces travaux, nous identifierons les limites de nos approches et nous discuterons d'améliorations futures, ainsi que d'avenues de recherche intéressantes.

8.1 Synthèse des travaux

Pour étudier le problème de planification de quarts de travail, nous avons utilisé différents modèles, tous basés sur des formulations issues de structures de programmation par contraintes exploitant les langages formels. Les langages formels permettent de formuler plusieurs règles dans la composition de quarts de travail de manière expressive et relativement naturelle.

Au chapitre 4, nous avons introduit deux modèles linéaires en nombres entiers 0-1 pouvant être résolus directement par un logiciel commercial et aborder les problèmes de planification de quarts multi-activités personnalisés. Le premier est construit à partir d'un automate définissant un langage régulier. À partir d'un automate, nous générons un graphe en couches où chaque couche représente une période d'un horizon de planification et chaque arc, une activité de repos ou de travail. Chaque chemin représente un quart faisant partie du langage défini par l'automate associé. À partir de ce graphe, un modèle de flot à coût minimum basé sur des variables d'affectations binaires est généré pour chaque employé pour identifier les horaires engendrant un coût global optimal. Le deuxième modèle suggéré est aussi basé sur un langage, mais celui-ci est défini par une grammaire hors-contexte plutôt que par un automate. À partir d'une grammaire définissant les règles régissant la composition des quarts pour un problème donné, un graphe orienté acyclique est construit. Le graphe orienté

contient tous les arbres d'analyse associés à des mots (quarts) reconnus par la grammaire. Un modèle en nombres entiers basé sur des variables binaires est ensuite dérivé du graphe associé à chaque employé de manière à identifier les arbres d'analyse, ou en d'autres mots, les quarts résultant en un horaire complet optimal. Dans le contexte où nous les utilisons, les grammaires ont l'avantage de pouvoir capturer plus de règles que les automates, notamment en ce qui concerne les longueurs et les positions de sous-séquences dans un quart, comme par exemple lorsqu'on souhaite restreindre la longueur d'une séquence de travail ou forcer la position d'une pause. Bien que ces deux modèles représentent un outil de modélisation puissant dans le contexte de création de quarts, lorsque le nombre d'employés et l'horizon de planification grandissent, leurs tailles peuvent devenir trop grandes et en pratique, il peut être impossible de les résoudre directement.

Au chapitre 5, nous avons proposé un modèle implicite qui peut pallier les limites du modèle basé sur les grammaires introduit au chapitre 4 dans les cas où les employés sont considérés comme identiques. Comme précédemment, à partir d'une grammaire restreignant la composition des quarts, nous générons le graphe orienté acyclique associé comportant tous les quarts réalisables. Lorsque nous générons le modèle, plutôt que d'utiliser un arbre par employé disponible et des variables binaires spécifiant si une activité donnée fait partie du quart sélectionné à une période donnée, nous utilisons un seul graphe et des variables entières positives qui spécifient dans combien de quarts intervient une variable donnée. Une procédure est utilisée pour identifier les quarts à partir de la solution optimale implicite. La formulation implicite basée sur les grammaires bénéficie donc de l'expressivité de modélisation qu'offrent les grammaires, mais ne dépend pas du nombre d'employés en ce qui concerne sa taille et évite les problèmes de symétries issus des contextes où les employés sont identiques. Les résultats expérimentaux montrent que cette approche de modélisation génère des modèles faciles à résoudre directement par un logiciel commercial et est particulièrement intéressant en ce qui concerne les problèmes multi-activités. De plus, nous avons montré que ce modèle engendre la même valeur de relaxation linéaire que les principaux modèles existants dans la littérature.

Au chapitre 6, nous avons développé une méthode de résolution permettant de résoudre efficacement les problèmes de planification de quarts multi-activités person-

nalisés. Notre approche utilise la méthode de génération de colonnes et un branchement adapté pour résoudre un modèle de recouvrement pour ces cas de problèmes. À partir du modèle de recouvrement comportant un nombre restreint de variables pour chaque employé disponible, les nouvelles variables (colonnes) sont générées à partir de sous-problèmes basés sur le graphe produit d'une grammaire spécifiant les quarts pouvant être effectués par l'employé en question. Les sous-problèmes sont résolus à partir d'un algorithme de programmation dynamique qui identifie les colonnes de coût minimum. Ensuite, pour trouver une solution entière à partir d'une solution fractionnaire, nous avons développé une méthode de branchement adapté au problème et facilement intégrable dans notre sous-problème. Les résultats expérimentaux montrent que notre méthode permet d'aborder différents types de problèmes de planification de quarts multi-activités personnalisés. Il s'agit d'une méthode exacte qui bénéficie de la flexibilité de modélisation des grammaires en plus d'être adapté pour des problèmes de grande taille.

8.2 Limitations de la solution proposée

Dans la section précédente, nous avons identifié quelques limitations associées à nos modèles, notamment en ce qui concerne le comportement des modèles du chapitre 4 lorsque le nombre d'employés augmente. De plus, tout comme les modèles du chapitre 4, lorsque les employés sont identiques, l'approche du chapitre 6 est très sensible à la symétrie. Ceci dit, dans les cas où les employés sont identiques, le modèle implicite du chapitre 5 reste le mieux adapté.

Maintenant, il convient de soulever quelques faiblesses de la modélisation avec les grammaires. Premièrement, certaines contraintes peuvent difficilement être formulées par une grammaire. Notamment, dans un contexte où il serait possible à un employé de travailler quelques heures, puis d'être en repos pour une période indéterminée et finalement de revenir travailler pour quelques heures, il est impossible, avec les modèles introduits dans ce travail, de contraindre le nombre total d'heures travaillées par cet employé à l'aide d'une grammaire. Par contre, si la durée de la période de repos est fixée, le cas peut être capturé par une grammaire. Cette situation se retrouve aussi si nous souhaitons planifier une semaine complète de travail sans préciser a priori les heures de début et de fin des quarts. Dans ce cas, borner le nombre total d'heures travaillées ou le nombre d'heures où l'employé travaille à une activité de travail donnée

est impossible. Pour tenir compte de ces cas, il est possible d'utiliser les modèles du chapitre 4 avec des contraintes supplémentaires. Ceci dit, cette limitation n'est pas unique à nos approches, plusieurs approches basées sur des réseaux sont aussi obligées d'ajouter des contraintes supplémentaires pour gérer ce type de cas.

Finalement, le graphe orienté acyclique généré à partir d'une grammaire peut être de très grande taille si l'horizon de planification comporte plusieurs périodes et être très dense si la grammaire comporte plusieurs productions. Les modèles en nombres entiers qui en dérivent peuvent devenir trop grands pour être résolus directement et l'algorithme de programmation dynamique présenté au chapitre 6 peut devenir plus lent à résoudre.

8.3 Améliorations et avenues de recherche futures

Pour pallier les problèmes où le nombre de périodes travaillées ou le nombre de périodes travaillées à une activité donnée doivent être comptabilisés et bornés, il serait intéressant d'avoir un concept de ressources pouvant être intégré à l'algorithme de programmation dynamique basée sur les grammaires suggéré au Chapitre 6. Une ressource pourrait être associée au nombre total de périodes travaillées et être accumulée de manière à être prise en compte lors d'un choix de noeud dans l'algorithme.

Comme avenues de recherche futures, il serait très bon de considérer les problèmes de planification de quarts de travail multi-activités avec affectations de tâches non interruptibles. Plusieurs contraintes peuvent être associées au placement de ces tâches dans les quarts de travail, comme un nombre d'employés requis et les compétences nécessaires pour l'effectuer. La tâche peut être contrainte à être effectuée dans une journée donnée ou dans une fenêtre de temps précise. Intégrer cette dimension à nos modèles serait un atout très intéressant.

L'utilisation de modélisations et méthodes basés sur les grammaires dans la planification intégrée de quarts et de cycles de travail semble aussi une extension naturelle à nos travaux. En particulier, il serait intéressant de voir si ce type d'approches pourrait être utilisé pour aborder les cas où les cycles de travail ne peuvent être énumérés a priori, comme c'est le cas dans des environnements où une grande flexibilité de planification est requise.

Références

ADDOU, I. et SOUMIS, F. (2007). Bechtold-Jacobs generalized model for shift scheduling with extraordinary overlap. *Annals of Operations Research*, 155, 177–205.

AHUJA, R., MAGNANTI, T. et ORLIN, J. (1993). *Network Flows.* Prentice Hall.

APT, K. (2003). *Principles of Constraint Programming.* Cambridge University Press.

AYKIN, T. (1996). Optimal shift scheduling with multiple break windows. *Management Science*, 42, 591–602.

AYKIN, T. (1998). A composite branch and cut algorithm for optimal shift scheduling with multiple breaks and break windows. *Journal of the Operational Research Society*, 49, 603–615.

BALAKRISHNAN, A. et WONG, R. (1990). Model for the rotating workforce scheduling problem. *Networks*, 20, 25–42.

BARNHART, C., HANE, C. et VANCE, P. (2000). Using branch-and-price-and-cut to solve origin-destination integer multicommodity flow problems. *Operations Research*, 48, 318–326.

BEAULIEU, H., FERLAND, J. A., GENDRON, B. et MICHELON, P. (2000). A mathematical programming approach for scheduling physicians in the emergency room. *Health Care Management Science*, 3, 193–200.

BECHTOLD, S. et JACOBS, L. (1990). Implicit modeling of flexible break assignment in optimal shift scheduling. *Management Science*, 36, 1339–1351.

BECHTOLD, S. et JACOBS, L. (1996). The equivalence of general set-covering and implicit integer programming formulations for shift scheduling. *Naval Research Logistics*, 43, 233–249.

BÖEHMER, M. et GRÜENER, J. (2003). Tour scheduling and task assignment of a heterogeneous work force : A model formulation and opl-implementation. Rapport technique, Universität der Bundeswehr Hamburg.

BOUCHARD, M. (2004). Optimisation des pauses dans le problème de fabrication des horaires avec quarts de travail. Mémoire de maîtrise, École Polytechnique de Montréal.

ÇEZIK, T., GÜNLÜK, O. et LUSS., H. (1999). An integer programming model for the weekly tour scheduling problem. *Naval Research Logistic*, 48.

COCKE, J. et SCHWARTZ, J. T. (1970). Programming languages and their compilers : Preliminary notes. Rapport technique, Courant Institute of Mathematical Sciences, New York University.

CÔTÉ, M.-C., GENDRON, B., QUIMPER, C.-G. et ROUSSEAU, L.-M. (2009). Formal languages for integer programming modeling of shift scheduling problem. *Constraints*, doi :10.1007/s10601-009-9083-2.

CÔTÉ, M.-C., GENDRON, B. et ROUSSEAU, L.-M. (2007). Modeling the regular constraint with integer programming. *Fourth International Conference on Integration of AI and OR Techniques in Constraint Programming (CP-AI-OR 07)*, 4510, 29–43.

CÔTÉ, M.-C., GENDRON, B. et ROUSSEAU, L.-M. (2010). Grammar-based integer programming models for multi-activity shift scheduling. *Submitted to Management Science*.

CRAINIC, T., FRANGIONI, A., GENDRON, B. et GUERTIN, F. (2009). Oobb : An object-oriented library for parallel branch-and-bound. Presented at the CORS/INFORMS International Conference, Toronto, Canada, June 14-17 2009.

DANTZIG, G. (1954). A comment on Edie's traffic delay at toll booths. *Operations Research 2*, 339–341.

DANTZIG, G. et WOLFE, P. (1960). Decomposition principle for linear programs. *Operations Research*, 8, 101–111.

DEMASSEY, S., PESANT, G. et ROUSSEAU, L.-M. (2005). Constraint programming based column generation for employee timetabling. *Proc. 2th Int. Conf. on Intergretion of AI and OR techniques in Constraint Programming for Combinatorial Optimization Problems - CPAIOR'05, Springer-Verlag LNCS*, 3524, 140–154.

DEMASSEY, S., PESANT, G. et ROUSSEAU, L.-M. (2006). A cost-regular based hybrid column generation approach. *Constraints*, 11, 315–333.

DESAULNIERS, G., DESROSIERS, J. et SOLOMON, M. M. (2005). *Column Generation.* Springer.

EDIE, L. (1954). Traffic delays at toll booths. *Journal of the Operations Research Society of America*, 2, 107–138.

ERNST, A., HOURIGAN, P., KRISHNAMOORTHY, M., MILLS, G., NOTT, H. et SIER, D. (1999). Rostering ambulance officers. *Proceedings of the 15th National Conference of the Australian Society for Operations Research, Gold Coast*, 470–481.

ERNST, A., JIANG, H., KRISHNAMOORTHY, M., OWENS, B. et SIER, D. (2004a). An annotated bibliography of personnel scheduling and rostering. *Annals of Operations Research*, 127, 21–144.

ERNST, A., JIANG, H., KRISHNAMOORTHY, M. et SIER, D. (2004b). Staff scheduling and rostering : A review of applications, methods and models. *European Journal of Operational Research*, 153, 3–27.

FELICI, G. et GENTILE, C. (2004). A polyhedral approach for the staff rostering problem. *Management Science*, 50, 381–393.

GABALLA, A. et PEARCE, W. (1979). Telephone sales manpower planning at qantas. *Interfaces*, 9, 1–9.

HENDERSON, W. et BERRY, W. (1976). Heuristic methods for telephone operator shift scheduling. *Management Science*, 22, 1372–1380.

HOPCROFT, J. E., MOTWANI, R. et ULLMAN, J. D. (2001). Introduction to automata theory, languages and computation. *Addison Wesley*.

KADIOGLU, S. et SELLMANN, M. (2008). Efficient context-free grammar constraints. *Proceedings of the 23rd National Conference on Artificial Intelligence*. 310–316.

KASAMI, T. (1965). An efficient recognition and syntax-analysis algorithm for context-free languages. Rapport technique AFCRL-65-758, Air Force Cambridge Research Lab, Bedford, MA.

KNIGHTON, S. et COCHRAN, J. (2005). A network-based mathematical programming approach to optimal rostering of continuous heterogeneous workforces. Rapport technique, Department of Industrial Engineering Fulton School of Engineering, Arizona State University, USA.

LAPORTE, G., NOBERT, Y. et BIRON, J. (1980). Rotating schedules. *European Journal of Operational Research*, 4, 24–30.

LASDON, L. S. (1970). *Optimization Theory for Large Systems*. The Macmillan Company.

LEQUY, Q., BOUCHARD, M., DESAULNIERS, G. et SOUMIS, F. (2009). Assigning multiple activities to work shifts. Rapport technique, Les Cahiers du GERAD G-2009-86, HEC Montreal, Montreal, Canada.

LOUCKS, J. et JACOBS, F. (1991). Tour scheduling and task assignment of a heterogeneous work force : a heuristic approach. *Decision Sciences*, 22, 719–739.

LÜBBECKE, M. E. et DESROSIERS, J. (2005). Selected topics in column generation. *Operations Research*, 53, 1007–1023.

MEHROTRA, A., MURTHY, K. et TRICK, M. (2000). Optimal shift scheduling : A branch-and-price approach. *Naval Research Logistics*, 47, 185–200.

MENANA, J. et DEMASSEY, S. (2009). Sequencing and counting with the multicost-regular constraint. *Proc. 6th Int. Conf. on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems - CPAIOR'09, Springer-Verlag LNCS 5547*. 178–192.

MILLAR, H. et KIRAGU, M. (1998). Cyclic and non-cyclic scheduling of 12 h shift nurses by network programming. *European Journal of Operational Research*, 104, 582–592.

MILLER, H., PIERSKALLA, W. et RATH, G. (1976). Nurse scheduling using mathematical programming. *Operations Research*, 24, 857–870.

MINOUX, M. (1989). *Programmation mathématique, théorie et algorithmes*. Paris, Dunod.

MOONDRA, S. (1976). A linear programming model for work force scheduling for banks. *Journal of Bank Research*, 6, 299–301.

MORRIS, J. et SHOWALTER, M. (1983). Simple approaches to shift, days-off and tour scheduling problems. *Management Science*, 29, 942–959.

OMARI, Z. (2002). Attribution des activités aux employés travaillant sur des quarts. Mémoire de maîtrise, École Polytechnique de Montréal.

PESANT, G. (2004). A regular membership constraint for finite sequences of variables. *Proceedings of the Tenth International Conference on Principles and Practice of Constraint Programming (CP 2004)*. 153–167.

PESANT, G., QUIMPER, C.-G., ROUSSEAU, L.-M. et SELLMANN, M. (2009). The polytope of context-free grammar constraints. *Proc. of CPAIOR'09, Springer-Verlag LNCS*, 5547, 223–232.

QUIMPER, C.-G. et ROUSSEAU, L.-M. (2009). A large neighbourhood search approach to the multi-activity shift scheduling problem. *Journal of Heuristics*, doi :10.1007/s10732-009-9106-6.

QUIMPER, C.-G. et WALSH, T. (2006). Global grammar constraints. *Proceedings of the Twelfth International Conference on Principles and Practice of Constraint Programming (CP 2006)*. 751–755.

QUIMPER, C.-G. et WALSH, T. (2007). Decomposing global grammar constraint. *Proceedings of the Thirteenth International Conference on Principles and Practice of Constraint Programming (CP 2007)*. 590–604.

REKIK, M. (2006). Construction d'horaires de travail dans des environnements hautement flexibles. Thèse de doctorat, École Polytechnique de Montréal.

REKIK, M., CORDEAU, J.-F. et SOUMIS, F. (2004). Using Benders decomposition to implicitly model tour scheduling. *Annals of Operations Research*, 128, 111–133.

REKIK, M., CORDEAU, J.-F. et SOUMIS, F. (2010). Implicit shift scheduling with multiple breaks and work stretch duration restrictions. *Journal of Scheduling*, 13, 49–75.

RITZMAN, L., KRAJEWSKI, L. et SHOWALTER, M. (1976). The disaggregation of aggregate manpower plans. *Management Science*, 22, 1204–1214.

SEGAL, M. (1974). The operator-scheduling problem : A network-flow approach. *Operations Research*, 22, 808–823.

SELLMANN, M. (2006). The theory of grammar constraints. *Proceedings of the Twelfth International Conference on Principles and Practice of Constraint Programming (CP 2006)*. 530–544.

SODHI, M. (2003). A flexible, fast, and optimal modeling approach applied to crew rostering at London Underground. *Annals of Operations Research*, 127, 259–281.

THOMPSON, G. (1995a). Improved implicit optimal modelling of the shift scheduling problem. *Management Science*, 41, 595–607.

VATRI, E. (2001). Intégration de la génération de quart de travail et de l'attribution d'activités. Mémoire de maîtrise, École Polytechnique de Montréal.

WARNER, D. (1976). Scheduling nursing personnel according to nursing preference : A mathematical programming approach. *Operations Research*, 24, 842–856.

YOUNGER, D. H. (1967). Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10, 189–208.

Annexe A

Schedule Construction

In the following, $V(N)$ is initialized to the value of the variable associated with node N in the implicit solution. $c_l(N)$ and $c_r(N)$ are the left and right children of and-node N. $c(N)$ is the set of children of or-node N. *schedule* is the schedule resulting from the algorithm, i.e., the shift assigned to each employee. L is the set of leaves in the DAG.

Data: Solution from an implicit grammar model
Result: Detailed schedule
Stack $K = \emptyset$;
Array $schedule[V(O^S_{1n}), n]$;
$e = 0$;
while $V(O^S_{1n}) > 0$ **do**
  $V(O^S_{1n}) = V(O^S_{1n}) - 1$;
  Choose $N \in \left\{c(V(O^S_{1n})) \mid V(N) > 0\right\}$;
  Push N on K;
  while $K \neq \emptyset$ **do**
    Pop N from K;
    $V(N) = V(N) - 1$;
    if $c_l(N) \in L$, $c_l(N)$ *corresponds to* O^j_{i1} **then**
      $schedule[e, i] = j$;
    else
      Choose $N_l \in \{c(c_l(N)) \mid V(N_l) > 0\}$;
      Push N_l on K;
      Choose $N_r \in \{c(c_r(N)) \mid V(N_r) > 0\}$;
      Push N_r on K;
    end
  end
  $e = e + 1$;
end

Algorithm 2: Extracting detailed schedules from an implicit grammar solution